Engenharia de Harness

Um livro vivo sobre o scaffolding que envolve agentes de IA

Gilsiley Henrique Darú · com co-autoria de IA (Claude, Anthropic)
v0.77.0 · gerado em 8 de agosto de 2026
DOI 10.5281/zenodo.21632412 · harness.ghdaru.com.br

Abertura

00 — Introdução

🕒 estado da arte 2026-07 · revisão 2026-08-01 · 📖 ~7 min de leitura

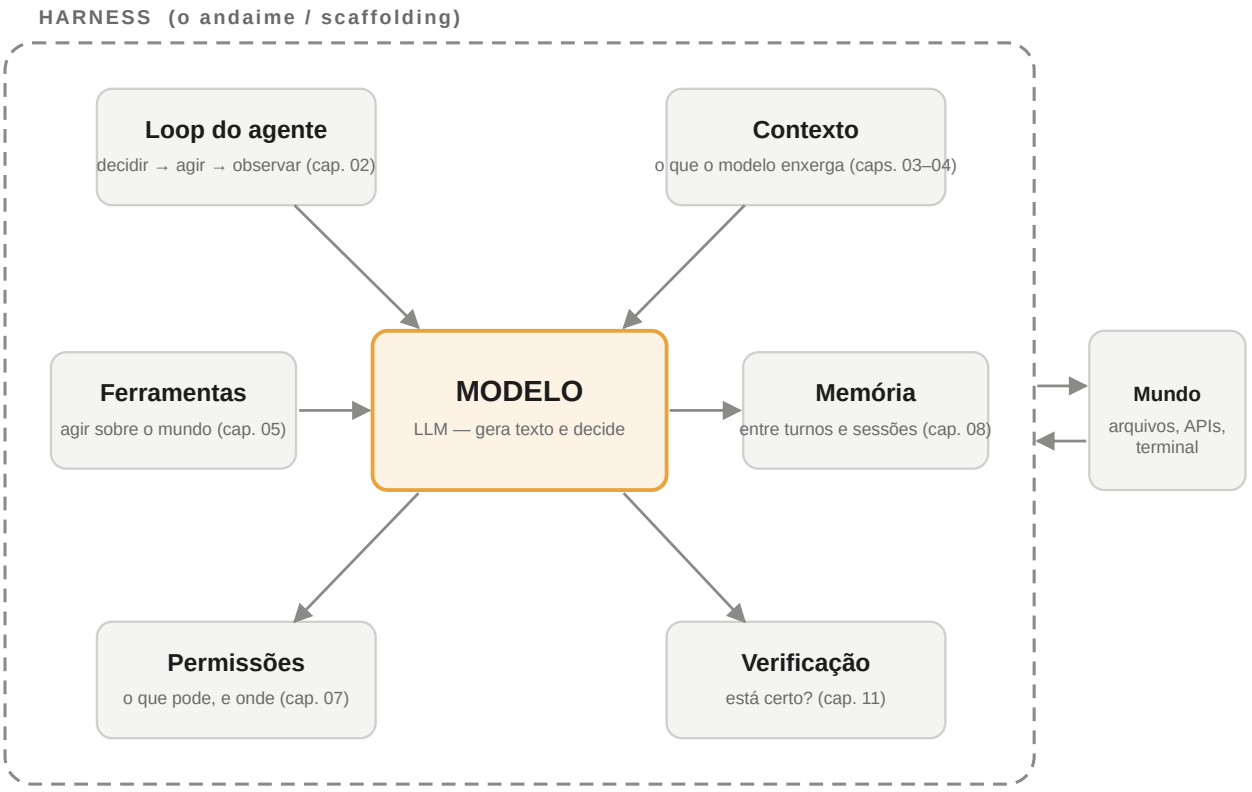
Agente = modelo + harness

Comece por uma pergunta que qualquer pessoa que já usou um chat de IA consegue fazer: por que o ChatGPT *responde* sobre o seu problema, mas não *resolve* o seu problema? Ele explica como corrigir o bug — mas não abre o arquivo, não roda o teste, não confere se funcionou. A resposta curta: um chat é só o **modelo**. Para o modelo *agir* — mexer em arquivos, executar comandos, verificar o próprio trabalho e parar na hora certa — é preciso construir uma estrutura inteira em volta dele. Essa estrutura é o assunto deste livro.

Quando um agente de IA resolve uma tarefa real — corrigir um bug, migrar um módulo, responder com base em dezenas de arquivos — duas coisas distintas estão trabalhando. A primeira é o **modelo**: a rede que lê contexto e decide o próximo passo. A segunda é tudo o que está em volta dele: quem monta o contexto que ele lê, quem executa as ferramentas que ele invoca, quem decide o que ele pode ou não fazer, quem lembra o que aconteceu ontem, quem verifica se o resultado está certo. Esse "tudo em volta" é o **harness** — em tradução livre, o arreio, o andaime, o *scaffolding*.

A fórmula que organiza este livro é simples:

> **agente = modelo + harness**



O modelo no centro; o harness — o andaime — em volta. Cada bloco é um capítulo deste livro.

O modelo é intercambiável e melhora a cada geração. O harness é engenharia de software clássica — e é nele que a maioria dos agentes falha ou tem sucesso. Dois produtos usando exatamente o mesmo modelo entregam resultados radicalmente diferentes conforme a qualidade do harness: como o contexto chega ao modelo, quais ferramentas ele tem, como os erros retornam, o que acontece quando a **janela de contexto** (o limite de texto que o modelo consegue "enxergar" de uma vez) acaba.

Engenharia de harness é a disciplina de projetar esse scaffolding: entrega de contexto, interfaces de ferramentas, artefatos de planejamento, loops de verificação, sistemas de memória e sandboxes.

Por que um livro — e por que agora

Entre 2024 e 2026, os harnesses de agentes de código deixaram de ser experimentos e viraram uma categoria de produto: Claude Code, Codex CLI, Gemini CLI, opencode, Aider, Cline, Goose, OpenHands e dezenas de outros. O mais notável não é a quantidade, mas a **convergência**: projetos independentes, em linguagens diferentes, chegaram às mesmas soluções — arquivos de contexto hierárquicos, compactação em camadas, plan mode como modo de permissão, hooks de ciclo de vida, MCP (Model Context Protocol) como padrão de integração.

Quando implementações independentes convergem, existe uma disciplina por trás. Este livro documenta essa disciplina.

O método: ler código, não marketing

Este livro é empírico. Cada capítulo trata de uma funcionalidade do harness (o loop, o contexto, a compactação, as permissões...) e é escrito a partir da leitura do código-fonte de harnesses reais de código aberto. A regra editorial mais importante do projeto:

> Afirmações sobre um harness exigem **evidência**: o caminho do arquivo no código-fonte onde a funcionalidade está implementada.

READMEs prometem; código entrega. Vários projetos anunciam dimensões que o código não tem — a exigência de evidência é o que separa avaliação de marketing.

Nota de autoria e método

Por transparência — e coerência com a regra de evidência acima — este livro é **co-escrito com um agente de IA** (Claude Code, da Anthropic) operando sob **autoria, curadoria e responsabilidade humanas**. O agente executa a pesquisa, a redação e o ciclo de produção; o autor humano define o escopo, decide, **verifica cada fonte** e responde pelo conteúdo. Seguindo as políticas editoriais de autoria (ICMJE, COPE, *Nature*, *Science*), a IA **não** é listada como autora — não pode ser responsável — e seu uso é divulgado aqui, na abertura.

Isso não é um detalhe: um livro sobre a disciplina de instrumentar bem os agentes de IA usa essa mesma disciplina para se escrever, e a expõe. O método completo — pesquisa dupla verificada por busca cruzada, ciclo spec-driven, revisão e datação — está documentado no Guia Editorial §6, com um *survey* das metodologias de escrita tradicionais e da era-IA que o fundamentam.

Como ler este livro — três portas de entrada

O livro foi escrito para ser denso; esta seção existe para que a densidade não seja uma parede. Escolha a sua porta:

- **Se você está chegando agora** (usou chats de IA, mas nunca construiu um agente): leia 00 → 01 → 02 em sequência, sem pressa, usando o Glossário como apoio — toda sigla do livro está lá, por extenso e explicada (na versão online, basta passar o mouse sobre a sigla). Depois do 02, os capítulos 03–13 podem ser lidos em qualquer ordem: cada um é autocontido e abre definindo o próprio problema.
- **Se você já opera um agente** (usa Claude Code, Codex, Cursor ou similares e quer entender o que há por dentro): a **Leitura executiva** ao fim de cada capítulo é o seu atalho — o estado da arte da dimensão em um parágrafo, com a seção "o que roubar". Vá direto aos capítulos do seu interesse e desça ao corpo quando quiser a evidência.
- **Se você constrói harnesses**: o livro inteiro é seu, incluindo os Apêndices A (evidência por repositório, com caminhos de arquivo), o Benchmark e as duas trilhas práticas — o **harness-zero** (construção didática, uma feature por etapa) e o **harness-um** (a implementação de referência completa, apêndice próprio).

Estrutura do livro

- **Fundamentos** (capítulo 01): as definições formais, os artigos canônicos e a taxonomia de problemas que organiza tudo o que vem depois.

- **Capítulos 02–13**: uma funcionalidade por capítulo. Cada um define o problema, apresenta os padrões de implementação conhecidos e mostra, com evidência, como cada harness estudado implementa.
- **Convergências e tendências** (capítulo 14): o que a indústria já padronizou, onde ainda há divergência real, e a "cláusula de expiração" — a tese de que todo componente de harness existe porque o modelo ainda não faz aquilo sozinho, e deve ser desenhado sabendo que um dia será desnecessário.
- **Capítulos 15–17**: as fronteiras — o harness embutido em produto (15), o harness que aprende com o uso (16) e a camada de protocolos que une o ecossistema (17).
- **Benchmark** (`benchmark/`): a seção empírica — avaliações padronizadas, por dimensão, com notas 0–3 e evidência, de cada harness estudado, mais o comparativo consolidado.

Os harnesses do estudo

O estudo cobre, até esta edição, **vinte e um sistemas de código aberto**, avaliados por leitura sistemática de código em cinco arquétipos (o método está no capítulo 01, §6):

- **Harnesses de código** — opencode, gemini-cli, OpenHarness, Codex CLI, Goose, Aider, OpenHands, Grok Build, Pi, Kimi Code e Prime Agent;
- **Agentes pessoais self-hosted** — OpenClaw, Hermes Agent, IronClaw, ohmo;
- **Agentes organizacionais** — QM;
- **Harnesses embutidos** — n8n (nó AI Agent);
- **Frameworks** — LangGraph, CrewAI, OpenAI Agents SDK (Software Development Kit), Software Agent SDK.

Cada um foi escolhido por representar um *arquétipo* diferente (lógica de replicação, não amostragem): produto maduro agnóstico de provedor (opencode), regime de controle de big tech (gemini-cli), port didático legível (OpenHarness), sandbox-first (Codex CLI), MCP-nativo (Goose), context-first (Aider), cultura de eval acadêmica (OpenHands), agente da organização inteira com o loop trocável (QM), e assim por diante.

A lista completa — com **origem, versão, fork e commit exatos lidos** em cada avaliação, e o link para a análise e o diagnóstico de cada um — está no **Apêndice — O estudo**. O placar consolidado por dimensão está no Comparativo.

Como referencial teórico e para explorar o ecossistema além do corpus, soma-se a coleção viva **Awesome Harness Engineering** (~426 recursos organizados por problema, na mesma organização deste livro) — de onde vêm a definição de harness usada no capítulo 01 e a taxonomia que estrutura os capítulos.

01 — Fundamentos

🕒 estado da arte 2026-07 · revisão 2026-08-01 · 📖 ~13 min de leitura

Este capítulo fixa o vocabulário, a **origem** e o **método** do livro. Antes de comparar harnesses (capítulos 02–13) é preciso responder três perguntas que a primeira edição deixou em aberto: *o que é* um harness, *de onde ele veio* (e o que havia antes), e *com que rigor* este livro o estuda.

1. O que é um harness (definição)

A definição de trabalho vem da lista curada awesome-harness-engineering:

> **Engenharia de harness** é a disciplina de projetar o *scaffolding* — **andaime** ou estrutura de suporte — que envolve um agente de IA (entrega de contexto, interfaces de ferramentas, artefatos de planejamento, loops de verificação, sistemas de memória e sandboxes) e determina se ele tem sucesso ou falha em tarefas reais.

Com o princípio orientador:

> O foco é o *harness*, não o modelo. Cada componente existe porque o modelo não consegue fazê-lo sozinho — e os melhores harnesses são projetados sabendo que esses componentes se tornarão desnecessários conforme os modelos melhoram.

Note o termo central: **scaffolding** (andaime). É a metáfora do livro — a estrutura provisória erguida em volta de algo em construção, que sustenta o trabalho e depois é removida. Guarde a palavra: ela reaparece no subtítulo, no título de cada parte e na §8 (a cláusula de expiração).

> **Para quem está chegando agora — uma imagem que sustenta o livro inteiro.** Pense no modelo como um profissional brilhante no primeiro dia de trabalho numa empresa que ele não conhece: capaz, mas sem mesa, sem acesso aos sistemas, sem saber as regras da casa — e com memória que zera a cada conversa. O harness é tudo que a empresa monta em volta dele: o dossiê do projeto que ele lê ao chegar (contexto, cap. 03), as ferramentas na bancada (cap. 05), o crachá que define onde pode entrar (permissões, cap. 07), o caderno de anotações que sobrevive ao fim do expediente (memória, cap. 08), o supervisor que revisa a entrega antes de ela sair (verificação, cap. 11) — e o expediente em si, o ritmo de trabalhar-conferir-continuar (o loop, cap. 02). Quando os capítulos ficarem técnicos, volte a esta imagem: cada dimensão do livro é uma peça desse escritório.

2. O que havia antes — e por que não eram agentes

"Software que age por você" é uma ideia antiga. As gerações anteriores, porém, resolviam o problema **sem um modelo de linguagem no centro do laço de decisão** — e é isso que as separa de um agente:

- **Sistemas especialistas** (anos 1980): regras `if-then` escritas à mão. Automatizavam decisões, mas não interpretavam objetivos em linguagem natural nem se recuperavam de exceções não previstas.
- **RPA — Robotic Process Automation** (UiPath, Automation Anywhere): robôs que repetem cliques e digitações por *script* fixo. Frágeis a qualquer mudança de tela; sem objetivo, sem recuperação.
- **Chatbots** de intenção (de ELIZA às árvores de diálogo): produziam texto, mas **não executavam ações** no mundo.
- **Assistentes de código como autocomplete**: o **GitHub Copilot** (technical preview em jun/2021), movido pelo modelo **OpenAI Codex** (descendente do GPT-3 ajustado em código), sugeria a próxima linha *dentro do editor* — sem plano, sem ferramentas, sem laço de verificação.

Nenhum deles tinha as **quatro peças** que hoje definem um harness (§4). Faltava-lhes autonomia orientada a objetivos e a capacidade de agir sobre o ambiente **e corrigir o próprio rumo**.

3. Como chegamos aqui — a linhagem técnica

A passagem de "modelo que responde" para "agente que age" foi construída em camadas, cada uma removendo um obstáculo:

1. **Raciocínio explícito.** O *Chain-of-Thought* (Wei et al., 2022) mostrou que pedir ao modelo para "pensar passo a passo" melhora tarefas de raciocínio.
2. **O loop.** O marco decisivo foi **ReAct — *Synergizing Reasoning and Acting in Language Models*** (Yao et al., arXiv:2210.03629, out/2022; ICLR 2023), que intercalou **Pensamento → Ação → Observação**: o modelo raciocina, chama uma ferramenta, observa o resultado e continua. Esse ciclo é o esqueleto de praticamente todo harness moderno (capítulo 02).
3. **A chamada de ferramentas.** Faltava um modo confiável de o modelo *invocar* ferramentas — resolvido quando a OpenAI lançou o **function calling** (jun/2023): o modelo emite JSON estruturado para acionar funções (capítulo 05).
4. **A onda autônoma — e sua lição.** Com raciocínio + ação + ferramentas, veio 2023: **AutoGPT** (Significant Gravitas, mar/2023) e **BabyAGI** (Yohei Nakajima, abr/2023) — loops que se decompunham em subtarefas e se executavam sozinhos. "Falharam" no sentido prático (entravam em círculos, gastavam tokens, perdiam o fio) porque tinham *o loop* mas **não** as outras três peças: gestão de contexto, ferramentas bem projetadas e controle. A lição fundadora da disciplina nasce aí: **o modelo sozinho não basta; o andaime em volta é que decide o sucesso.**
5. **O amadurecimento — os CLIs de código.** As quatro peças foram embutidas em ferramentas de terminal ligadas ao sistema de arquivos e ao Git: **Aider** (Paul Gauthier, abr/2023), **Claude Code** (Anthropic, research preview em fev/2025), **OpenAI Codex CLI** (open source, abr/2025), além de projetos como **Cline**, **OpenHands** e **SWE-agent**.
6. **A padronização.** Com agentes proliferando, vieram os protocolos: o **Model Context Protocol (MCP)**, aberto pela Anthropic (nov/2024), padronizou a conexão a ferramentas e dados (capítulo 06); o **AGENTS.md** consolidou-se como "README para agentes"; o **Agent2Agent (A2A (Agent-to-Agent))** (Google, abr/2025; depois doado à Linux Foundation) endereçou a comunicação *entre* agentes (capítulo 17).

Linha do tempo (marcos): 1980s sistemas especialistas · 2000s–2010s RPA e chatbots · **jun/2021** Copilot (autocomplete) · **out/2022** ReAct · **mar–abr/2023** GPT-4, AutoGPT, BabyAGI, Aider · **jun/2023** function calling · **nov/2024** MCP · **fev/2025** Claude Code · **abr/2025** Codex CLI e A2A.

> **Nota de rigor.** "Codex" designa três coisas distintas — o *modelo* de 2021 (base do Copilot), a *linha de produto* Codex da OpenAI e o *Codex CLI* open source de 2025. O texto as mantém separadas. Datas e fontes desta seção estão na Bibliografia; itens ainda a verificar estão marcados lá.

4. A definição constitutiva: os quatro elementos

A literatura da disciplina converge numa definição do harness como uma **camada de runtime** com quatro elementos necessários e suficientes:

1. **Loop do agente** — o ciclo que alterna entre invocar o modelo e executar o que ele decidiu, até um critério de parada (cap. 02).
2. **Interface de ferramentas** — o contrato pelo qual o modelo age sobre o mundo: ler arquivos, rodar comandos, chamar APIs (cap. 05).
3. **Gestão de contexto** — a montagem, priorização e compressão do que o modelo enxerga a cada chamada (caps. 03–04).
4. **Mecanismos de controle** — permissões, aprovações, sandboxes e limites que restringem o que o agente pode fazer (cap. 07).

Um sistema sem qualquer um dos quatro **não é um harness completo**: um chatbot com ferramentas mas sem loop é um "function caller"; um loop sem controle é um incidente esperando acontecer; ferramentas sem gestão de contexto colapsam em tarefas longas. **Esta é a definição operacional que serve de teste de inclusão** do estudo (§5–6).

As quatro peças numa tarefa real. Peça a um agente: "o teste `test_login` está falhando, corrija". O que acontece, peça a peça: a **gestão de contexto** monta o que o modelo vai enxergar (as regras do projeto, a mensagem, talvez o arquivo do teste); o modelo lê e decide pedir uma ação — "rode o teste e me mostre o erro" — que a **interface de ferramentas** executa de verdade no terminal; o resultado volta, o modelo propõe editar um arquivo, e os **mecanismos de controle** decidem se essa edição pode acontecer direto ou se precisa da sua aprovação; aplicada a edição, o **loop** realimenta o modelo com o novo estado — teste passa? — e repete o ciclo até o critério de parada. Quatro peças, um turno de trabalho. Os capítulos 02–13 são este parágrafo em câmera lenta.

5. De onde vêm os harnesses deste estudo

O corpus é **de código aberto** (o Princípio II do livro: a fonte-base é o código) e se divide em cinco arquétipos — os mesmos do capítulo 00:

- **Harnesses de código** (opencode, gemini-cli, OpenHarness, Codex CLI, Goose, Aider, OpenHands, Grok Build, Pi, Kimi Code, Prime Agent): implementações de referência que juntam as quatro peças num executável.
- **Agentes pessoais self-hosted** (OpenClaw, Hermes Agent, IronClaw, ohmo): o harness a serviço de uma pessoa, com identidade, memória e canais próprios.
- **Agentes organizacionais** (QM): o harness a serviço de uma organização — escopos, permissões por audiência e auditoria como primitivas, com o loop do agente como motor trocável.
- **Harnesses embutidos** (n8n, nó AI Agent): o loop como componente dentro de um produto maior.
- **Frameworks** (LangGraph, CrewAI, OpenAI Agents SDK, Software Agent SDK): expõem loop, estado e ferramentas como primitivas programáveis.

O **teste de inclusão** é a definição da §4: entra quem tem *loop + ferramentas + gestão de contexto + controle*; ficam de fora bibliotecas de modelo puro e meros *wrappers* de uma ferramenta. A lista avaliada, com o repositório e o commit lido de cada um, está no Comparativo e no apêndice do estudo. Recursos consultáveis além do corpus estão na coleção viva Awesome Harness Engineering.

6. O método do estudo (rigor)

Este livro **lê o código-fonte de harnesses reais**, os compara por dimensões e depois **constrói um harness do zero**. Isso não é "opinião de engenheiro": é um desenho de pesquisa híbrido que se apoia em tradições metodológicas consolidadas. Explicitá-las converte o livro de coletânea de impressões em **estudo empírico auditável** — coerente com o Princípio I ("evidência acima de retórica").

Em linguagem simples, antes dos nomes técnicos: o método é (1) escolher sistemas que representem *tipos* diferentes de harness, não os mais famosos; (2) ler o código de cada um seguindo **o mesmo roteiro de perguntas**, anotando o arquivo exato que prova cada resposta; (3) dar notas por uma régua fixa e publicada, para que qualquer pessoa possa discordar olhando a mesma evidência; e (4) construir um harness do zero para testar se os padrões extraídos realmente se sustentam. Os parágrafos a seguir dão os nomes formais de cada uma dessas escolhas e de onde elas vêm — são a genealogia do rigor, e podem ser lidos em diagonal na primeira passada.

Duas fases, dois motores.

- **Fase 1 — descritiva/comparativa:** um **estudo de casos múltiplos** (Yin) apoiado em **Mining Software Repositories** (Hassan, 2008), tratando cada repositório como *dado primário*. A unidade de análise é **o código-fonte**, não o material de marketing nem o comportamento observado em uso.
- **Fase 2 — construtiva/prescritiva:** o `harness-zero` é um exercício de **Design Science Research** (Hevner et al., 2004; processo DSRM de Peffers et al., 2007): projetar e avaliar um artefato que instancia os princípios extraídos na Fase 1.

Como as dimensões viram medida. As dimensões de comparação descem pelo método **Goal–Question–Metric** (Basili, Caldiera & Rombach): para cada objetivo de harness (contexto, ferramentas, permissões, memória, verificação, loop, orquestração) formulam-se perguntas e, para cada pergunta, **indicadores observáveis no código** (ex.: existe mecanismo de compactação? qual a granularidade do modelo de permissões? há camada de verificação pós-ação?).

Seleção por replicação, não por amostragem. Os casos são escolhidos pela **lógica de replicação** de Yin — *literal* (espera-se o mesmo padrão) ou *teórica* (espera-se diferença previsível) — com critérios explícitos: código aberto e inspecionável na data de corte; pertencer à classe "harness" (§4); relevância de adoção **ou** singularidade arquitetural; diversidade de arquétipos (§5). Para cada caso registram-se **URL, commit/tag e data de leitura**.

Codificação e síntese. A leitura segue um protocolo comum a todos os casos (Runeson & Höst, 2009), combinando codificação indutiva inspirada em *grounded theory* (Stol, Ralph & Fitzgerald, 2016) na descoberta das dimensões e *análise de conteúdo* (Hsieh & Shannon, 2005) com grade fixa na pontuação. A síntese comparativa é uma **feature analysis** no estilo **DESMET** (Kitchenham, Linkman & Law, 1997), na tradição do *benchmarking* como motor de progresso científico (Sim, Easterbrook & Holt, 2003).

Ameaças à validade (taxonomia de Cook & Campbell, 1979, adaptada a estudo de caso):

Tipo	Ameaça	Mitigação declarada
Constructo	as "dimensões" não capturarem o que define um harness	derivação por GQM; definições operacionais publicadas
Interna	atribuir a "boa prática" o que é acaso histórico do projeto	protocolo único; cada afirmação rastreada a trecho/commit
Externa / **obsolescência**	não generalizar; o campo muda em meses	seleção por arquétipos; **data de corte + commits fixos**; a **cláusula de expiração** (§8) é a mitigação declarada, não um enfeite
Conclusão	tratar notas qualitativas como métrica exata	escala e critérios explícitos (DESMET); sem agregação numérica espúria

Assim cada afirmação do livro remete a **um dado no repositório** e a **um procedimento nomeado**. O detalhamento operacional está no Comparativo e no template de avaliação; as referências, na Bibliografia.

7. Taxonomia por problema

Convenção herdada do referencial: organizar a disciplina **pelo problema resolvido, não por fabricante ou modelo**. É a taxonomia que estrutura os capítulos:

Problema	Capítulo
Como o ciclo de decisão-ação funciona e quando para	02 — Loop do Agente
O que o modelo enxerga e como isso é montado	03 — Entrega de Contexto
O que fazer quando a janela de contexto acaba	04 — Compactação
Como o modelo age sobre o mundo	05 — Design de Ferramentas
Como integrar capacidades externas de forma padronizada	06 — MCP
O que o agente pode fazer, e onde	07 — Permissões e Sandboxing
O que persiste entre turnos e entre sessões	08 — Memória e Estado
Como trabalho grande vira passos verificáveis	09 — Planejamento
Como distribuir trabalho entre múltiplos agentes	10 — Subagentes e Orquestração
Como saber se o agente (e o harness) funcionam	11 — Verificação e Evals
Como terceiros estendem o harness	12 — Extensibilidade
Por onde humanos e sistemas usam o agente	13 — Interfaces

8. A cláusula de expiração

A tese mais importante — e menos praticada — da disciplina: **todo componente de harness é uma prótese temporária.** A compactação existe porque janelas de contexto são finitas; o *plan mode* existe porque modelos agem precipitadamente; o *policy engine* existe porque modelos não são confiáveis com comandos destrutivos. Cada premissa tem prazo de validade.

O corolário prático: todo componente deveria documentar **qual melhoria de capacidade do modelo o tornaria desnecessário**. Harnesses que não fazem isso acumulam *scaffolding* morto — complexidade que sobrevive à limitação que a justificava. Como visto na §6, essa cláusula é também a **mitigação declarada** da ameaça de obsolescência: o livro se assume datado. Voltamos a ela no capítulo 14.

9. Artefatos operacionais

A disciplina produziu artefatos-padrão que reaparecem, com variações, em quase todos os harnesses estudados:

- **Arquivo de instruções de projeto** (`AGENTS.md` / `CLAUDE.md` / `GEMINI.md`): regras, convenções e limites que o agente lê antes de qualquer tarefa. Fronteiras claras superam restrições vagas.

- **Artefato de plano** (`PLAN.md`): criado no início da tarefa e atualizado durante a execução, com marcos verificáveis e fronteiras de escopo.
- **Log de implementação** (`IMPLEMENT.md`): registro *append-only* de decisões e desvios do plano.
- **Checklist de harness** (`HARNESS_CHECKLIST.md`): revisão pré-produção cobrindo instruções, ferramentas, contexto, planejamento, permissões e verificação — com a tabela de expiração da §8.

Esses artefatos são o embrião do nosso instrumento de avaliação (ver `benchmark/template/HARNESS_EVAL.md`).

---

*As fontes deste capítulo (históricas e metodológicas) estão consolidadas na Bibliografia, separando as **confirmadas** das que ainda pedem verificação — fiel ao Princípio I.*

Capítulos por funcionalidade

02 — Loop do Agente

🕒 estado da arte 2026-07 · revisão 2026-08-01 · 📖 ~9 min de leitura

Objetivos de aprendizagem

1. **Explicar** o ciclo prompt → decisão → ferramenta → resultado e o critério de parada estrutural;
2. **Comparar** os dois contratos de terminação da indústria (ausência de tool call × `output_type` satisfeito);
3. **Implementar** um loop com freios (turnos, orçamento) e trace observável (etapa 1 do harness-zero);
4. **Projetar** retry em duas camadas (dentro do passo × replay do loop) e reconhecer o que exige idempotência;
5. **Avaliar** a durabilidade de um loop real (o que sobrevive a um crash?).

O problema

O loop é o coração do harness: envia contexto ao modelo, recebe uma decisão (texto e/ou **tool calls** — pedidos estruturados de ação: "execute tal ferramenta com tais argumentos"), executa, realimenta e repete — até que alguém decida parar.

Um turno completo, em câmera lenta. Você digita: "o teste `test_login` falhou, corrija". O que o loop faz:

1. Monta o contexto (regras do projeto + sua mensagem) e **chama o modelo**;
2. O modelo não responde com texto — responde com uma tool call: `executar_shell("pytest test_login")`;
3. O harness **executa de verdade** e devolve a saída (o traceback do erro) ao modelo, como se fosse uma nova mensagem;
4. O modelo agora *viu* o erro e emite outra tool call: `editar_arquivo("auth.py", …)`;
5. O harness executa (talvez pedindo sua aprovação — cap. 07) e devolve o resultado;
6. Nova chamada ao modelo, que pede o teste de novo; desta vez passa;
7. O modelo responde **só com texto** ("corrigido: era o cookie expirado") — e é *isso* que encerra o turno: **sem tool call, o loop para**.

Sete passos, três chamadas ao modelo, duas execuções reais. Todo o resto deste capítulo são as perguntas difíceis escondidas nesse ciclo: quem decide parar (e se o modelo nunca parar?), como os erros voltam, o que acontece quando o processo morre no passo 5, quanto isso pode custar. As perguntas de projeto: quem decide parar? como os resultados e erros voltam? o que acontece quando dá errado? o loop sobrevive a um reinício?

Fundamentos científicos

- **ReAct** (arXiv 2210.03629) é o paper seminal: intercalar raciocínio e ação com feedback do ambiente supera raciocínio puro — é a justificativa científica de o loop existir.
- O survey de **frameworks de raciocínio agêntico** (arXiv 2508.17692) sistematiza as variantes do ciclo (ReAct, plan-and-act, reflexão), útil como mapa do território.
- A fronteira treinada: surveys de **agentic search com RL** (arXiv 2510.16724) mostram o loop deixando de ser só orquestração e virando objeto de treinamento — quando o modelo é treinado *no* loop, parte do harness migra para os pesos.

(Bibliografia completa: `livro/bibliografia.md`.)

Fontes da indústria

- **How the agent loop works** (Claude Agent SDK (Software Development Kit), docs): o loop canônico em 5 estágios; "turno" termina **quando o modelo responde sem tool calls**; e o detalhe mais moderno — terminação como **estado tipado** (`success`, `error_max_turns`, `error_max_budget_usd`…): sucesso e esgotamento de limite são caminhos de código distintos e obrigatórios. Inclui `max_budget_usd` **propagado a subagentes** e a compactação como evento observável do loop (`compact_boundary`).
- **Loop engineering** (Claude blog): o vendor batiza a disciplina e dá a taxonomia por eixos (como dispara, como para, que primitivo usa) — com a regra de projeto citável: *se você não consegue escrever a verificação, o loop não está pronto para existir*.
- **Building Effective AI Agents** (Anthropic): a distinção fundadora workflow × agente e o padrão **evaluator-optimizer** — parada semântica (qualidade atingida) com um juiz separado.

- **Running agents** (OpenAI Agents SDK): o contrato alternativo — parada quando o agente produz o **`output_type` declarado** (validável), com `MaxTurnsExceeded` tipado.
- **LoopAgent** (Google ADK): só duas formas de parar — `max_iterations` ou um sub-agente juiz emitindo `escalate=True` — o loop burro separado do juiz endereçável.
- **Durable AI Loops** (Restate) e Inngest: o loop como **workflow de longa duração** — cada passo journalado, falha = replay do último passo concluído; retry vira duas categorias (backoff dentro do passo × replay do loop), com idempotência obrigatória em tools mutantes.
- **Consulte também**: a coleção viva Awesome Harness Engineering — Agent Loop reúne mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

1. Parada virou contrato multi-eixo

O critério estrutural (sem tool call) continua universal, mas sozinho é ingênuo. O contrato moderno combina: limite de turnos; **teto de orçamento em dinheiro** (a novidade real de 2025–26, já propagando a subagentes); *subtype* tipado de terminação; e, no contrato alternativo do Agents SDK, **parada por tipo de saída** — que transforma "acabou?" em validação verificável. Sobre isso, dois refinamentos medidos no benchmark: o **next-speaker check** do gemini-cli (uma inferência barata decide se o modelo continua sozinho) e o veto de término — hooks `Stop` que podem **recusar o fim do turno** e reinjetar feedback (software-agent-sdk; o verify-on-stop do Hermes é o mesmo princípio como nudge).

2. Anti-runaway: do contador ao detector

Todo loop maduro tem `MAX_TURNS`; os melhores têm detecção de repetição — `LoopDetectionService` (gemini-cli), `RepetitionInspector` (Goose), stuck detector com estados `stalled/stuck` (software-agent-sdk, OpenClaw). A técnica de campo (hash de `tool+args` em janela deslizante) circula entre praticantes mas não tem doc de vendor — citável como prática, não como norma.

3. Durabilidade virou propriedade do loop, não da infra

O consenso 2026: journaling por passo + replay. No benchmark: rollouts jsonl recuperáveis (Codex), inbox durável de prompts com eventos replayáveis por cursor (opencode V2), event log append-only com retomada por diretório (software-agent-sdk) e — o desenho mais radical — o executor que **retorna apenas referências duráveis** e nunca muta estado, com um applier validando evidência antes de aplicar (IronClaw). Corolário para tools: idempotência deixa de ser virtude e vira requisito.

4. O loop não é o perímetro

A lição arquitetural mais importante da rodada 2 (IronClaw): *"the loop is intentionally not the security perimeter"* — o loop pede efeitos por portas; quem decide é o kernel. Mesmo fora do contexto de segurança, a separação política (quando parar/confirmar/desistir — `Conversation.run()`) × mecânica (view → LLM → dispatch — `Agent.step()`) do software-agent-sdk é o corte limpo que permite trocar o motor mantendo o loop.

> **LEITURA EXECUTIVA**
>
> O que está mais moderno: terminação tipada com orçamento em dólares; juiz separado e endereçável (evaluator-optimizer/escalate) em vez de heurística no prompt; durabilidade por journaling/replay; e a separação política×mecânica. **O que roubar:** `ResultMessage.subtype` tipado; budget propagado a subagentes; hooks Stop com poder de veto; o LoopExit por referências duráveis.

Mão na massa — harness-zero, etapa 1

A etapa 1 (`harness-zero/etapas/01-loop/`) implementa o núcleo em ~30 linhas: parada estrutural, `MAX_TURNS` como freio, erros de tool voltando **como texto** para o modelo decidir, e trace das ações visível no chat. Exercícios de extensão: (a) adicione um subtype de terminação (`success` × `max_turns`); (b) adicione um orçamento de custo estimado e o terceiro subtype.

Verificação

1. Por que "o modelo respondeu sem tool calls" é um bom default de parada — e por que é insuficiente sozinho? (Contrato multi-eixo.)
2. Seu agente chamou a mesma tool com os mesmos argumentos 5 vezes seguidas. Liste duas defesas de naturezas diferentes. (Detector de repetição × teto de orçamento.)
3. O processo morreu no meio do turno 7. O que o seu loop precisa ter persistido para retomar sem repetir efeitos colaterais? (Journaling + idempotência.)

Apêndice A — Como cada repositório trata o loop

> Evidência por harness, com paths — complementação online, expandida a cada rodada.

opencode (rodada 1)

`packages/opencode/src/session/processor.ts`: resposta consumida como `Stream` do Effect (`Stream.tap(handleEvent)` → `takeUntil(needsCompaction)` → `runDrain`); veredito explícito `continue | stop | compact`; retry por provedor (`SessionRetry.policy`); V2 (`CONTEXT.md`): inbox durável e eventos replayáveis com cursores.

gemini-cli (rodada 1)

`packages/core/src/core/client.ts` (`MAX_TURNS=100`) + `turn.ts`; **next-speaker check** (`utils/nextSpeakerChecker.ts`: mini-prompt `{reasoning, next_speaker}` re-invoca o stream se `model`); `LoopDetectionService`; separação core/cli limpa.

OpenHarness (rodada 1)

`src/openharness/engine/query.py` (`run_query`): `while` async até `max_turns` ou sem tool-uses; **paralelismo quando todas as tools do turno são read-only** (`asyncio.gather`); PreToolUse → permissão → execução → PostToolUse por chamada; retry com backoff e cost tracking.

Codex CLI (rodada 2)

`core/src/session/turn.rs` (`run_turn`, 2.581 linhas) sobre `SessionTask` trait (Regular/Review/Compact/UserShell); streaming SSE (Server-Sent Events) **e WebSocket com fallback WS→HTTPS**; `CancellationToken` hierárquico; cada turno persistido em rollout jsonl; sem detector de repetição explícito (mitigado por budgets).

Goose (rodada 2)

`crates/goose/src/agents/agent.rs` (`reply` → `BoxStream<AgentEvent>`): dois níveis de retry (transiente por provedor + `RetryManager` de recipe com `SuccessCheck` que reseta a conversa); `DEFAULT_MAX_TURNS=1000`; `RepetitionInspector`; `MAX_EMPTY_TURN_RETRIES=3`.

OpenClaw (rodada 2)

`src/system-agent/agent-turn.ts` + `gateway/agent-*.ts`: runs serializados por *session lane* com write-lock file-based inter-processo; três streams de eventos (lifecycle/assistant/tool); watchdogs `stalled/stuck`; hooks duplos (Gateway + plugins).

Hermes (rodada 2)

`agent/conversation_loop.py` (~6.5k linhas) com fases separadas (turn_context/tool_executor/turn_finalizer); `iteration_budget`; **interrupt-and-redirect** (`/steer` drenado pré-API e pós-tool); nudges para respostas vazias; reparo de alternância de papéis; **verify-on-stop nudge**.

IronClaw (rodada 2) ⭐

`crates/ironclaw_agent_loop`: pipeline de estágios selados (input → prompt → model → capability → gate/checkpoint → stop), cada estágio uma strategy privada; o executor devolve um `LoopExit` contendo **apenas referências duráveis** — nunca muta estado — e o `LoopExitApplier` valida evidência host-owned antes de aplicar (tese explícita da arquitetura: *"the loop is intentionally not the security perimeter"*). Estado resumível por checkpoints; máquina de estados Queued→Running→Blocked→Completed com leases/heartbeats; "one active run per canonical thread".

Aider (rodada 2)

`aider/coders/base_coder.py`: não é um loop de tool-calling — é REPL de chat + edição direta. O único mecanismo iterativo é a **reflexão** (`reflected_message`, máx. 3): arquivos pedidos fora do chat, erros de linter ou testes falhando disparam nova rodada, sempre com confirmação humana. Auto-correção reativa por design, não autonomia.

OpenHands/Canvas (rodada 2)

`app_server/event/`: o event-stream persiste cada `Event` como JSON por conversa (paginação, filtros, export de trajetória) — mas o loop ação/observação roda no `openhands-agent-server` (SDK); o app consome eventos, não os gera. O núcleo está no software-agent-sdk (abaixo).

ohmo (rodada 2.5)

Loop herdado do `QueryEngine` do OpenHarness; o que é próprio: **pool multi-sessão** (`ohmo/gateway/runtime.py`: um `RuntimeBundle` por `session_key`, recriado quando o cwd muda) e **interrupção real por mensagem nova** (`bridge.py`: cada mensagem é uma asyncio.Task; mensagem nova da mesma sessão cancela a anterior) — poucos concorrentes cancelam corretamente.

n8n (rodada 2)

A V2 usa o `AgentExecutor` clássico do LangChain (`maxIterations` default 10); a **V3** mantém o `createToolCallingAgent` só para *decidir* — as tool calls viram `EngineRequest` devolvidos ao **motor de workflow do n8n**, que agenda os nós-tool e reentra com `EngineResponse` (`ToolsAgent/V3/helpers/runAgent.ts`). O n8n reinternalizou o loop de execução: decisão do framework, execução do engine.

Frameworks (rodada frameworks) — quatro respostas à mesma pergunta

LangGraph: a primitiva real é **Pregel/BSP** (supersteps + channels + reducers), com retry/cache/timeout por nó — e o agente pronto (`create_react_agent`) formalmente deprecado (migrou para `langchain.agents`). **OpenAI Agents SDK (Software Development Kit)**: loop explícito em `run.py` (output_type termina · handoff troca agente · `max_turns` com handlers), sobre um `AgentRunner` substituível. **CrewAI**: executor **100% próprio, zero LangChain** (`crew_agent_executor.py`), com dispatch duplo — tool-calling nativo ou fallback ReAct com `json_repair`. **software-agent-sdk**: `LocalConversation.run()` (política: parar, confirmar, desistir) separado de `Agent.step()` (mecânica stateless view → LLM → dispatch), event log append-only com `View` derivada e hooks `Stop` com poder de **veto** sobre o término.

03 — Entrega de Contexto

🕒 estado da arte 2026-07 · revisão 2026-07-25 · 📖 ~11 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Explicar** por que contexto é um orçamento gerenciado em runtime, não um depósito (e o que é *context rot*);
2. **Compor** um system prompt em camadas ordenadas por volatilidade (cache-aware);
3. **Projetar** uma cascata de arquivos de contexto (global → projeto → pacote → pessoal) com precedência declarada;
4. **Implementar** o montador de contexto do harness-zero (etapa 3) com um arquivo de regras de projeto;
5. **Avaliar** um arquivo AGENTS.md real contra as práticas de autoria (enxuto, comandos executáveis, crescido por evidência de falha).

O problema

O modelo só sabe o que o harness mostra. "Entrega de contexto" é a engenharia de decidir **o que** entra em cada chamada — system prompt, regras do projeto, estado do ambiente, memórias, instruções de servidores externos — **em que ordem**, e **como isso muda** no meio de uma conversa sem quebrar o cache do provedor nem confundir o modelo.

Sub-problemas clássicos: onde vivem as regras do projeto e como são descobertas; se o prompt de sistema deve variar por modelo; como informar mudanças de estado mid-conversation sem invalidar o prefixo cacheado.

Fundamentos científicos

- **Contexto degrada com posição e com volume** — *Lost in the Middle* (arXiv 2307.03172): a informação no meio de contextos longos é mal utilizada. Consequência de projeto: o que importa vai para as bordas (system prompt no início; a tarefa atual no fim), e "mandar tudo" é anti-padrão com base empírica.
- **Context engineering como disciplina** — o survey arXiv 2507.13334 sistematiza a área (RAG, memória, tool-integrated reasoning) e legitima o termo que a indústria adotou.
- **Menos contexto, agentes melhores** — arXiv 2606.10209 mede em agentes de longa duração o que a Anthropic chama de context rot: curadoria agressiva supera janelas cheias.

(Bibliografia completa: `livro/bibliografia.md`.)

Fontes da indústria

- **Effective context engineering for AI agents** (Anthropic Engineering): batiza a sucessão da prompt engineering — o trabalho é **curar o conjunto ótimo de tokens em tempo de inferência**; nomeia *context rot* como fato de engenharia. Decisão: a janela é orçamento, e a meta é o menor conjunto de tokens de alto sinal.
- **Prompt caching** (docs oficiais) + **Lessons from building Claude Code: prompt caching is everything**: cache é **por prefixo** — a ordem de montagem do contexto é decisão de custo. O relato do Claude Code lista os invalidadores clássicos (timestamp no topo, request ID na lista de tools, reserialização do histórico) e trata **cache hit rate como métrica de primeira classe do harness** (~59% de redução de input billable).
- **AGENTS.md** + **Agentic AI Foundation**: o "README para agentes" foi **doado à Linux Foundation (dez/2025)** com OpenAI, Anthropic e Block como co-fundadores; 60k+ projetos. Decisão: contexto por arquivo de repositório virou infraestrutura neutra e portável — investir nesse pipeline é seguro.
- **How Claude remembers your project** (docs): formaliza a **cascata** global → projeto → local, com o arquivo mais próximo vencendo e o pessoal fora do versionamento.
- **AGENTS.md Field Guide 2026** (praticante): autoria — começar com ~30 linhas, teto ~150–200 na raiz, comandos exatos antes de prosa, aninhar por pacote em monorepo, e **crescer só por evidência de falha reincidente do agente** (o erro comum é tratá-lo como documentação).

- **Consulte também**: a coleção viva Awesome Harness Engineering — Context Delivery & Compaction reúne mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

1. Contexto é orçamento gerenciado — e a recuperação virou just-in-time

O consenso moderno inverteu o instinto de "quanto mais contexto, melhor": o harness administra ativamente a janela (poda por regra, awareness de quanto resta, recuperação sob demanda). As duas materializações mais avançadas do benchmark: o **repo-map** do Aider (o modelo "enxerga" a estrutura de um repositório inteiro num orçamento de ~1k tokens, via tree-sitter + PageRank personalizado — recuperação estática just-in-time sem nenhum agente explorador) e os **hints incrementais por subdiretório** do Goose (regras carregadas conforme o agente navega, não tudo de antemão).

2. Estabilidade de prefixo virou requisito arquitetural

Cache-awareness deixou de ser otimização e reorganizou a montagem do contexto: camadas ordenadas por volatilidade, serialização determinística, zero conteúdo volátil no topo. As formalizações mais rigorosas medidas: os **Context Epochs** do opencode (o prefixo como baseline imutável de cache, com mudanças de estado entregues só em fronteiras seguras de turno) e o **prompt em três camadas explícitas** do Hermes (`stable` → `context` → `volatile`, desenhado declaradamente para maximizar prefix-cache — inclusive no fork de curadoria de skills, que herda o prefixo do pai para economizar ~26%).

3. O arquivo de regras padronizou — e virou cascata

A fragmentação AGENTS/CLAUDE/GEMINI.md do início da disciplina está resolvida por governança neutra (Linux Foundation): AGENTS.md é o formato portável, lido nativamente por Codex, Goose, opencode, OpenClaw, Hermes, Aider e dezenas de outros, com os nomes proprietários virando alias. O padrão maduro é a **cascata com precedência declarada** (global → projeto → pacote → pessoal; o mais próximo vence; o pessoal gitignored), `@imports` para composição (gemini-cli) e — a prática de autoria que separa arquivos úteis de documentação morta — crescer **por evidência de falha**, como código.

4. As fronteiras novas

Três movimentos recentes que ainda não viraram consenso: **prompt por família de modelo** (opencode com ~10 variantes; Codex levando ao extremo com instruções **server-driven** — o backend entrega o prompt-base por modelo, com até "personalidade" configurável); **separação persona × regras** (a contribuição da categoria de agentes pessoais: `SOUL.md` para voz/identidade separado do `AGENTS.md` operacional — OpenClaw, Hermes, ohmo); e **contexto com classe de confiança** (IronClaw: conteúdo pessoal/injetado viaja em "prompt envelopes" com trust class preservada — a entrega de contexto encontrando a segurança do cap. 07).

> **O contraponto: o harness mínimo (Pi)** — *adendo da rodada ext-1, 2026-07-31.* Enquanto este capítulo descreve montadores de contexto cada vez mais ricos, o Pi (Earendil/Zechner, ~54k estrelas) aposta na direção oposta: system prompt base **medido em ~460 tokens**, derivado do tool set (cada ferramenta contribui seu snippet; guidelines entram só se a ferramenta correspondente está ativa), e skills anunciadas **só por nome+descrição** — o corpo é carregado pelo próprio modelo via `read` quando a tarefa pede (a divulgação progressiva levada ao limite: nem tool de skill existe). A honestidade editorial exige as duas ressalvas que a leitura de código revelou: (1) o mesmo montador concatena os `AGENTS.md` da cascata **sem orçamento** — no próprio repo do Pi isso adiciona ~2.700 tokens, seis vezes o slogan; a minimalidade é do harness, não do contexto; (2) o minimalismo não é ausência de engenharia — a compactação do Pi é a mais completa do corpus (ver avaliação). A aposta subjacente é falsificável e vale acompanhar: **modelos melhores precisariam de menos harness** — se for verdade, parte deste capítulo expira; se a janela continuar cara, a falta de orçamento cobra juros. É o experimento de controle que faltava ao corpus.

LEITURA EXECUTIVA

O que está mais moderno: orçamento + just-in-time (não volume), prefixo estável como requisito (com cache hit rate como SLI), AGENTS.md em cascata sob governança neutra, e as três fronteiras (prompt por modelo/server-driven, persona separada, trust class). O contraponto minimalista (Pi, rodada ext-1) mostra o outro extremo do espectro: prompt de ~460 tokens derivado do tool set — e prova que a tensão orçamento×riqueza segue aberta. **O que roubar:** repo-map como alternativa barata à exploração; as 3 camadas por

> volatilidade do Hermes; a disciplina "cresce por falha reincidente" na autoria de AGENTS.md; do Pi, o snippet de prompt acoplado à definição da ferramenta (prompt e tool set nunca dessincronizam).

Mão na massa — harness-zero, etapa 3

Na etapa 3 você constrói o montador de contexto do harness-zero: system prompt em camadas ordenadas por volatilidade (identidade → ambiente → regras do projeto → memória → tarefa), descoberta de um `AGENTS.md` na raiz do projeto-alvo, e um teste que prova a **estabilidade do prefixo** entre dois turnos consecutivos (mesmos bytes até a última mensagem). Exercício de completude: a função de descoberta em cascata vem esqueletada; você implementa a precedência.

Verificação

1. Por que um timestamp no topo do system prompt é caro — e onde ele deveria ficar? (Cache por prefixo + mid-conversation updates.)
2. Seu agente ignora uma convenção do projeto de forma reincidente. Qual é a resposta certa segundo a prática de autoria moderna — e qual é a errada? (Adicionar a regra ao AGENTS.md por evidência × despejar documentação.)
3. Um harness quer informar ao modelo que a data mudou no meio de uma conversa longa. Descreva duas estratégias com custos de cache diferentes. (Epochs/fronteiras de turno × reescrever o prefixo.)

Apêndice A — Como cada repositório trata a entrega de contexto

> Evidência por harness, com paths — complementação online, expandida a cada rodada do benchmark.

opencode (rodada 1) — álgebra tipada e Context Epochs

`packages/opencode/src/session/system.ts` monta environment + skills + instruções MCP (Model Context Protocol); **~10 prompts por família de modelo** em `session/prompt/*.txt` (anthropic, gpt, codex, gemini, kimi, beast...), selecionados por substring do model id; `AGENTS.md` globais/ascendentes agregados por `session/instruction.ts`. A V2 (`CONTEXT.md`) formaliza o contexto como álgebra de "Context Sources" com snapshots, **Context Epochs** (baseline de cache) e mensagens de sistema mid-conversation só em fronteiras seguras.

gemini-cli (rodada 1) — hierarquia com @imports

`prompts/promptProvider.ts` monta por modo/tools/modelo (snippets modernos × legados); `GEMINI.md` hierárquico (`memoryDiscovery.ts`: global → pais → subpastas) com `@imports` (`memoryImportProcessor.ts`) e `flattenMemory`; override total via `GEMINI_SYSTEM_MD`; injeção just-in-time (`tools/jit-context.ts`).

OpenHarness (rodada 1) — agregação com memória relevante

`src/openharness/prompts/context.py`: base + ambiente + `CLAUDE.md` + **memórias selecionadas por relevância** (`memory/relevance.py`, com `usage.py` rastreando uso) + skills + contexto de repo ativo; `-s/--append-system-prompt` na CLI.

Codex CLI (rodada 2) — AGENTS.md central + prompts server-driven

`core/src/agents_md.rs`: descoberta hierárquica com merge do project-root ao cwd; system prompt **varia por modelo e vem do backend** (`ModelInfo.base_instructions` via `models-manager`, com template e `Personality::Friendly/Pragmatic`); contexto ambiental via `WorldState`.

Goose (rodada 2) — hints incrementais e hardening

`SystemPromptBuilder` com override + extras; hints multi-arquivo (**`.goosehints` E `AGENTS.md`**, `CLAUDE.md` via config) respeitando `.gitignore`; **`SubdirectoryHintTracker`** carrega hints de subdiretório conforme o agente navega; sanitização anti prompt-injection de tags Unicode; "top of mind" por turno.

Aider (rodada 2) — o repo-map ⭐

`aider/repomap.py`: tags de definição/referência via tree-sitter (queries `.scm` por linguagem) → grafo arquivo→arquivo → **PageRank personalizado** (chat files e idents mencionados enviesam o ranking; multiplicadores ×10/×50/×0.1) → renderização sob orçamento com busca binária (~1024 tokens; `map_mul_no_files=8` sem arquivos no chat) → cache por mtime. O caminho context-first inteiro em um arquivo.

OpenHands/Canvas (rodada 2) — skills organizacionais

`app_conversation/skill_loader.py`: skills auto-descobertas de repositórios convencionais **`owner/.openhands` e `owner/.agents`** em todas as organizações do usuário (GitHub/GitLab/Azure), com KeywordTrigger/TaskTrigger e marketplace — contexto de time versionado e carregado para todos os membros.

OpenClaw (rodada 2) — workspace de identidade com orçamentos

`buildAgentSystemPrompt` injeta `SOUL.md` (persona), `AGENTS.md` (regras), `USER.md`, `IDENTITY.md`, `TOOLS.md`, `MEMORY.md`, `HEARTBEAT.md`, `BOOTSTRAP.md` — com orçamentos (20k chars/arquivo, 60k total) e truncamento marcado; contribuições provider-aware **acima/abaixo do cache boundary**.

Hermes (rodada 2) — três camadas por volatilidade ⭐

`agent/system_prompt.py` + `prompt_builder.py`: `stable` (identidade/SOUL.md + guidance + índice de skills) → `context` (AGENTS.md/.cursorrules do projeto) → `volatile` (memória, USER.md, timestamp) — desenho explícito para prefix-cache; persona migrável do OpenClaw.

IronClaw (rodada 2) — contexto como decisão de política

`LoopPromptPort` (crates/ironclaw_loop_host): resolve identidade, contexto pessoal (**opt-in por run profile, não por canal**), skills e segurança; conteúdo injetado/pessoal viaja em **prompt envelopes** com trust class inforjável — separação entre o que o loop pede e o que o host permite ver.

ohmo (rodada 2.5) — a versão mínima correta

`ohmo/prompts.py`: concatenação ordenada base → soul → identity → user → BOOTSTRAP → workspace → memória; decisão rigorosa `include_project_memory=False` (o agente pessoal não lê CLAUDE.md de projeto — testado).

Pi (rodada ext-1) — o prompt derivado do tool set ⭐

`core/system-prompt.ts`: base **medida em ~460 tokens**, montada dos `promptSnippet` das próprias tool definitions com dedup e guidelines condicionais ao conjunto ativo (desativou a tool, o prompt encolhe); skills anunciadas só como `<name/description/location>` e carregadas pelo modelo via `read` (bloco omitido se `read` não está ativa); cascata `AGENTS.md`/`CLAUDE.md` global→raiz→cwd com dedup de worktrees aninhadas (`resource-loader.ts`) — porém concatenada **sem orçamento** (ver caixa no corpo do capítulo); override total via `.pi/SYSTEM.md`.

n8n (rodada 2) — o mínimo do embutido

`ToolsAgent/common.ts`: `ChatPromptTemplate` com system message livre + histórico + binários ricos (imagens/PDF); sem arquivo de regras nem hierarquia — o contexto vem mapeado do workflow pelo autor.

Frameworks (rodada frameworks) — aberto por design

LangGraph e Agents SDK (Software Development Kit) deixam a montagem por conta do dev (instructions estáticas ou callable); CrewAI impõe role/goal/backstory como contexto estrutural; o software-agent-sdk dá preset Jinja com escape hatch documentado (`prompt_dir` + `_prompt_preset() -> None`).

04 — Compactação

🕒 estado da arte 2026-07 · revisão 2026-07-25 · 📖 ~13 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Explicar** por que a compactação existe e quais restrições ela equilibra (fidelidade × custo × cache);
2. **Comparar** as quatro camadas da escada de agressividade e **justificar** a ordem entre elas;
3. **Analisar** a implementação de compactação de um harness real e localizar suas escolhas na escada (Apêndice A como gabarito);
4. **Implementar** truncamento com preservação de bordas e sumarização com tail preservado (etapa 5 do harness-zero);
5. **Avaliar** quando uma compactação falhou (perda de decisão, de estado de arquivo ou de objetivo) — e **antecipar** o que muda quando o provedor compacta por você.

O problema

Toda conversa de agente cresce até não caber na janela de contexto do modelo. A compactação é o conjunto de estratégias para continuar trabalhando quando isso acontece — sem perder o que importa. É a dimensão onde os harnesses avaliados mais convergem: todos chegaram, independentemente, à mesma arquitetura em camadas.

As restrições em tensão:

- **Fidelidade**: o resumo não pode perder decisões, estado de arquivos ou o objetivo da tarefa.
- **Custo**: sumarizar via LLM (Large Language Model) é caro; truncar é barato mas destrutivo.
- **Cache**: compactar invalida o prefixo cacheado — deve acontecer o mínimo possível e em momentos controlados.

Fundamentos científicos

- **A janela não é uniforme** — *Lost in the Middle* (arXiv 2307.03172) mostrou que modelos usam melhor o início e o fim do contexto e degradam no meio. É a base empírica de duas práticas da escada: preservar o *tail* recente intacto e truncar outputs mantendo início+fim.
- **Contexto como memória virtual** — *MemGPT* (arXiv 2310.08560) formulou a analogia com sistemas operacionais: a janela é a "RAM", o armazenamento externo é o "disco", e o harness pagina entre eles. Trabalhos recentes levam a analogia ao limite literal (*demand paging*, arXiv 2603.09023).
- **Compactar é decisão de orçamento** — *ContextBudget* (arXiv 2604.01664) trata a gestão de contexto como alocação explícita por tipo de conteúdo — o que os produtos implementam como limiares e budgets.

(Bibliografia completa e status de validação: `livro/bibliografia.md`.)

Fontes da indústria

- **Compaction — Claude Platform Docs** (Anthropic, oficial): a compactação chegou **ao nível da API** (beta `compact-2026-01-12`) — o provedor sumariza automaticamente ao atingir o limiar configurado e devolve um "compaction block". É a confirmação de vendor da tendência central deste capítulo (ver Estado da arte).
- **Práticas de operação do Claude Code** (CometAPI, okhlopkov, hyperdev): a recomendação convergente dos praticantes é a mesma que os harnesses codificam — **o que precisa sobreviver à compactação não deve morar na conversa**: convenções vão para o arquivo de contexto (CLAUDE.md/AGENTS.md, reinjetado a cada sessão) e estado de progresso vai para arquivos que o agente relê depois do compact. A compactação define, por exclusão, o que merece persistência.
- **Consulte também**: a coleção viva Awesome Harness Engineering — Context Delivery & Compaction reúne mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

O padrão consolidado: a escada de agressividade

Os harnesses aplicam as estratégias em escada, da mais barata à mais cara — este é o consenso da indústria, verificado em todas as rodadas do benchmark:

1. **Truncar saídas de tools na origem** — limitar linhas/bytes antes de entrar no histórico, preservando início e fim (*Lost in the Middle* justifica as bordas). O refinamento moderno: **não descartar** — mover o conteúdo integral para arquivos referenciáveis (opencode) ou manter o bruto fora da view do modelo mas visível na UI (Goose).
2. **Prune / microcompact** — apagar o *conteúdo* de resultados de tools antigas (o modelo raramente relê um `cat` de 30 turnos atrás), mantendo o registro da chamada. Camadas intermediárias mais novas: *tool distillation* e *output masking* (gemini-cli).
3. **Sumarização via LLM (full compact)** — resumir a porção antiga preservando um tail intacto (tipicamente 20–30% ou um orçamento de 2k–20k tokens). O estado da arte tem três refinamentos: **resumo estruturado** com campos obrigatórios (intenção do usuário, tarefas pendentes, estado de código — Goose e software-agent-sdk), **modelo auxiliar barato** para o resumo (Hermes), e **flush de memória antes de compactar** — salvar notas duráveis antes de perder o contexto (OpenClaw).
4. **Disparo automático + caminho reativo** — gatilho por percentual da janela (50–90% conforme o projeto) e, cobrindo a falha, compactação **reativa** ao erro "prompt too long" da API (OpenHarness, OpenClaw).

As duas fronteiras modernas

1. Compactação auditável (tombstones). A implementação mais avançada medida no benchmark (condenser do software-agent-sdk) não muta o histórico: o log é append-only e o esquecimento é um *evento* (`Condensation`) — um tombstone, como em Cassandra/Kafka. A view do modelo é derivada aplicando os tombstones; nada se perde para auditoria, e invariantes formais (pareamento tool_call/result, atomicidade de batch) são **código testável**, com a distinção *hard/soft trigger*: se compactar agora violaria uma invariante, o gatilho suave espera o próximo turno; o duro força um reset explícito. Refinamento correlato: o **circuit-breaker de efetividade** (IronClaw) — comparar a estimativa pós-compactação contra baseline e detectar compactações que não estão funcionando.

2. A compactação está migrando para o provedor. (E o cache também vira contrato de protocolo: a spec MCP 2026-07-28 adicionou `ttlMs`/`cacheScope` às respostas de `tools/list` — o protocolo assumindo o que antes era heurística do harness.) Dois sinais independentes no mesmo ano: o Codex CLI implementa **compactação remota v2** (o backend compacta) e a Anthropic lançou **compaction na própria API** (docs, beta `compact-2026-01-12`). É a cláusula de expiração em movimento — mas com uma inversão interessante: em vez de o componente desaparecer quando o modelo melhora, ele **muda de dono** (do harness para a plataforma). O que resta ao harness quando o provedor compacta: decidir *o que proteger* (skills, estado de tarefa, arquivos de memória), *quando confiar* (auditoria de qualidade do resumo — o modo `safeguard` do OpenClaw antecipou isso) e o caminho reativo para provedores que não oferecem o serviço.

> **Adendo (2026-07-31, texto integral verificado): a terceira via — compactação aprendida no treino.** O preprint CompactionRL (Tsinghua/Z.AI, 06-jul-2026) propõe o passo seguinte da migração: treinar o modelo por RL **com a compactação dentro do loop** — "CompactionRL incorporates compaction into rollout collection, and reconstructs the agent context from a summary once context budget is exhausted" (§1); sumarização vira "a learned part of the model rather than an inference-time heuristic", com recompensa de nível de **tarefa**. Os números (Tabela 2, sempre contra o mesmo modelo *já com compactação de inferência*): GLM-4.5-Air **59,8 → 66,8** no SWE-bench Verified (+7,0) e +3,1 no Terminal-Bench 2.0; GLM-4.7-Flash **+5,5 e +6,8**. E o protocolo do experimento é exatamente a escada deste capítulo — limiar por orçamento restante, sumário estruturado por prompt fixo, **cauda preservada de k=2 passos** — ou seja, o paper valida a tríade e muda o *treino*, não a arquitetura. Três consequências: (1) o harness continua dono do *quando*, mas o *como sumarizar* começa a migrar para os pesos — descasamento harness ↔ modelo vira risco novo; (2) a limitação declarada é reveladora: "its gains do not consistently transfer to single-window evaluation when compaction is disabled. This indicates a train–test mismatch" — compactação treinada cria *acoplamento* (com compactação desligada, o GLM-4.7-Flash treinado chega a **piorar**, 47,5 → 43,7), o argumento mais forte até agora para o *contrato de compactação* explícito entre harness e modelo; (3) na contramão, a Tabela 1 devolve poder ao harness: fixado o executor, **trocar só o sumarizador** move o SWE-Verified de 49,0 a 55,5 (+6,5) — "compaction is a performance-critical decision process rather than a passive preprocessing step", e um sumarizador dedicado melhor **supera o auto-sumário**: escolher quem resume é decisão de harness, e das grandes.

A terceira fronteira: a compactação deixa de ser involuntária (rodada ext-4, 2026-08)

O lançamento do **Prime Agent** (Prime Intellect, ago/2026) veio com uma acusação direta a este capítulo: *"fixed tool-calling schemas and context compaction force the model to work around its own scaffolding instead of leveraging it"*. A leitura do código (avaliação completa) mostra que **a acusação é retórica e o código diz outra coisa** — e a diferença entre as duas é o achado.

A compactação **não foi eliminada nem enfraquecida**. O Prime Agent é construído sobre o Pi, e as 1.398 linhas de `core/compaction/` estão lá intactas — corte seguro, split turns, arquivos cumulativos, recuperação reativa de overflow —, ainda melhoradas com instruções customizadas e recálculo de `tokensBefore`. O que mudou é **quem manda**: `compact.run()` e `compact.status()` viraram chamáveis **pelo próprio agente** (`skills/compact/`), com um handler que **agenda em vez de executar** — executar na hora abortaria a célula do REPL que pediu a compactação — e que roda mesmo com a compactação automática desligada, sob doze casos de teste. Some-se a isso que o caminho do JSONL da sessão é injetado no system prompt: o histórico completo, **inclusive as compactações anteriores**, continua acessível por programa.

A ressalva a registrar neste capítulo é, portanto, precisa: **a compactação deixa de ser um evento involuntário do harness e passa a ser um mecanismo entre outros, disponível ao agente**. Ela ganha ainda um papel novo — virou gatilho de destilação, com `autoRefine.compact: true` por padrão: toda compactação é uma oportunidade de o agente extrair aprendizado do que está prestes a ser resumido.

O que a escada de agressividade não previa não é a sua obsolescência, mas a **inversão do controle**: até aqui, o harness compacta *no* agente; aqui, o agente compacta *a si mesmo*. A lacuna que a leitura encontrou é reveladora — o anúncio menciona um subagente atuando como coletor de lixo do REPL, e **não existe nada disso no código** (busca por `garbage`/`prune`/`evict` em `src/core`, `skills` e `prime-agent-runtime` não retorna nada). O contexto-como-variável resolve o acesso ao passado; **não** resolve o crescimento do namespace que ele mesmo cria.

LEITURA EXECUTIVA

A convergência na escada é quase total — o padrão está consolidado e um harness novo que não a implemente precisa justificar. As diferenças que restam são refinamentos de fidelidade (estruturar o resumo, auditar sua qualidade, nunca descartar) e a grande questão em aberto é de *arquitetura de mercado*: quanto da escada sobrevive no harness quando a plataforma oferece compaction como serviço — questão que o adendo acima agudiza: depois de migrar para o provedor, a compactação começa a migrar **para os pesos**. **O que roubar** hoje: tombstones sobre log append-only; memory-flush pré-compactação; resumo estruturado com IDs de tarefa preservados; circuit-breaker de efetividade; e — novo na ext-4 — **compactação chamável pelo agente que agenda em vez de executar**, mais o caminho do log da sessão no system prompt, que devolve o passado ao alcance do modelo sem gastar janela.

> **Ressalva de edição (2026-08-06).** Esta Leitura executiva foi confrontada na rodada ext-4 e **mantida**, com a qualificação da seção anterior: a escada continua sendo o padrão, mas a *autoridade* sobre quando aplicá-la começou a migrar para o agente. Se o padrão se repetir em outros harnesses, a síntese muda — e este parágrafo será reescrito, não emendado.

Mão na massa — harness-zero, etapa 5

Na etapa 5 do projeto (`harness-zero/`), você implementa a escada no seu próprio harness, nesta ordem: (1) truncamento de output de tool com preservação de início+fim; (2) prune de resultados de tools antigas além de um orçamento; (3) sumarização via LLM da cabeça do histórico, preservando o tail; (4) disparo automático por limiar de tokens estimados — com um **indicador visível no chat** quando a compactação acontece (a janela de observação do leitor). Exercício de completude: o esqueleto da função de prune vem pronto; você escreve a seleção do que proteger.

Verificação

1. Por que truncar outputs de tools **antes** de sumarizar via LLM, e não o contrário? (Custo e destrutividade — se precisar, releia a escada.)
2. Um harness sumarizou o histórico e o agente, no turno seguinte, reescreveu um arquivo que já estava correto. Qual informação a compactação provavelmente perdeu, e qual mecanismo do estado da arte previne isso? (Dica: resumo estruturado com

`CODE_STATE/CHANGES`.)

3. Seu provedor passou a oferecer compaction na API. Quais responsabilidades da escada você **transfere** e quais **mantém** no harness? (Conecte com "as duas fronteiras modernas".)

Apêndice A — Como cada repositório trata a compactação

> Evidência por harness, com paths — material de complementação (versão online), expandido a cada rodada do benchmark. Fonte-base do capítulo: o código destes repositórios.

opencode (rodada 1) — três mecanismos + arquivos gerenciados

`packages/opencode/src/session/compaction.ts` (+ `overflow.ts`, `summary.ts`): (a) sumarização automática em overflow com **agente dedicado `compaction`**, tail sob orçamento (`preserveRecentBudget`, 2k–8k tokens), novo Context Epoch e auto-continue opcional; (b) **prune** de trás para frente marcando `compacted` saídas de tools além de 40k tokens (`PRUNE_PROTECT`), protegendo skills; (c) truncamento na origem (`tool/truncate.ts`) preservando início+fim e movendo o texto completo para "Managed Tool Output Files".

gemini-cli (rodada 1) — compressão + destilação + mascaramento

`packages/core/src/context/chatCompressionService.ts`: dispara a 50% do limite (`DEFAULT_COMPRESSION_TOKEN_THRESHOLD = 0.5`), preserva os últimos 30% (`COMPRESSION_PRESERVE_THRESHOLD`), orçamento próprio para function responses (50k) e salvamento de outputs truncados. Camadas extras: `toolDistillationService.ts` e `toolOutputMaskingService.ts`. `/compress` manual, evento `ChatCompressed`, hooks `PreCompressTrigger`.

OpenHarness (rodada 1) — a tradução fiel do Claude Code

`src/openharness/services/compact/__init__.py` (1.725 linhas; docstring: "Faithfully translated from Claude Code's compaction system"): **microcompact** (limpa `COMPACTABLE_TOOLS`), **full compact** (resumo LLM), **auto-compact** (limiar) e compactação **reativa** a "prompt too long" (`_is_prompt_too_long_error`). Hooks `PRE_COMPACT`/`POST_COMPACT`; preserva task state e logs de canal.

Codex CLI (rodada 2) — local + remota v1/v2

`core/src/compact.rs`, `compact_remote_v2.rs`, `compact_token_budget.rs`: auto-compact a ~90% da janela; três estratégias — local (`SUMMARIZATION_PROMPT`) e **remota v1/v2** (o backend compacta, via `ResponsesStreamRequest::RemoteCompactionV2`, com retry próprio); janelas versionadas com prefill tracking; injeção controlada pré/mid-turn; `TruncationPolicy` para outputs.

Goose (rodada 2) — resumo estruturado + middle-out

`crates/goose/src/context_mgmt/mod.rs`: limiar 0.8 da janela; `StructuredSummary` (user_intent, files, pending_tasks, current_work); se a sumarização estoura, **remoção progressiva "middle-out"** de tool-responses (0→100%); **sumarização incremental de pares tool-call/response** em batches de 10 protegendo os N últimos; metadados de visibilidade preservam o bruto na UI; respeita `provider.manages_own_context()`.

OpenClaw (rodada 2) — safeguard + memory flush

`src/context-engine/` + `docs/concepts/compaction.md`: auto por limiar e reativa (reconhece dezenas de strings de erro de overflow de múltiplos provedores), split preservando pares tool-call/result; modo `safeguard` com **auditoria de qualidade do resumo**; **memory flush silencioso antes de compactar**; `keepRecentTokens` 20k; providers de compactação plugáveis; distinção compaction (semântica) × pruning (trim in-memory).

Hermes (rodada 2) — engine plugável + modelo auxiliar

`agent/context_engine.py` (interface `should_compact`/`compress`/`prune`) + `trajectory_compressor.py` (~1.6k linhas): sumarização de tool-responses antigas via **modelo auxiliar barato** (default Gemini Flash, até 50 requisições concorrentes); `/compress` manual; `/usage` e `/insights` expõem a janela.

IronClaw (rodada 2) — política pura + circuit-breaker

`crates/ironclaw_agent_loop/src/strategies/compaction.rs` (+ `active_task_compaction.rs`): a estratégia é **política pura** (retorna Skip ou o limite `drop_through_seq`; mutação só no host); `PromptContextTokenBudget` com `preserve_tail_tokens`; **circuit-breaker de efetividade** (compara estimativa pós-compactação contra `CompactionEffectivenessBaseline`); variante que preserva a tarefa ativa; o host rejeita compactar através de mensagens não-usuário.

software-agent-sdk (rodada frameworks) — tombstones + invariantes testáveis ⭐

`openhands-sdk/openhands/sdk/context/condenser/`: esquecimento por **tombstones** (`Condensation` event) sobre log append-only; disparo por três razões (REQUEST/TOKENS/EVENTS) com **hard/soft** (`condensation_requirement`) e `hard_context_reset()` para o caso patológico; `keep_first` + re-sumarização recursiva de sumários; prompt estruturado (`summarizing_prompt.j2`: USER_CONTEXT, TASK_TRACKING com IDs exatos, CODE_STATE, TESTS, CHANGES); invariantes em `context/view/properties/` (tool_call_matching, batch_atomicity...) **testadas contra LLMs reais** (`tests/integration/tests/c01..c05`); `pipeline_condenser` para compor.

Aider (rodada 2) — sumarização clássica bem-feita

`aider/history.py` (`ChatSummary`): mantém a cauda (~metade do orçamento), sumariza a cabeça via LLM com split após mensagem `assistant`, **recursivo** até profundidade 3, com lista de modelos com fallback.

n8n (rodada 2) — a ausência que confirma a categoria

Sem compactação no loop (`contextWindowLength` dos memory sub-nodes + `maxTokensFromMemory` apenas) — coerente com execuções curtas acionadas por evento; é o teto da categoria "harness embutido" para tarefas longas.

LangGraph / OpenAI Agents SDK / CrewAI (rodada frameworks) — a linha divisória

LangGraph: **zero suporte nativo** (uma docstring sugerindo `pre_model_hook`); Agents SDK (Software Development Kit): apenas `OpenAIResponsesCompactionSession` como session opcional; CrewAI: nada. A compactação é a dimensão que mais separa "framework" de "harness pronto".

05 — Design de Ferramentas

🕒 estado da arte 2026-07 · revisão 2026-07-25 · 📖 ~8 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Explicar** por que a descrição de uma tool é prompt engineering, não documentação de API;
2. **Derivar** o schema de uma tool a partir de tipos (e justificar por que ninguém mais escreve JSON Schema à mão);
3. **Comparar** os três regimes de escala — catálogo fixo, tool search com carregamento tardio, e code-as-action;
4. **Implementar** a `ToolPort` do harness-zero com schema derivado e erro-como-dado (etapa 2);
5. **Avaliar** quando usar tool calls individuais × código orquestrando tools em sandbox.

O problema

As ferramentas são as "mãos" do agente: o contrato pelo qual o modelo age sobre o mundo. Design de ferramentas é decidir **quais** existem, **como** seus parâmetros são descritos ao modelo, **como** os resultados (e erros) retornam, e **quando** cada uma está disponível. Uma tool mal descrita gera chamadas erradas; um arsenal grande demais dilui a atenção do modelo *e* estoura o orçamento de contexto antes de qualquer trabalho útil; um arsenal pequeno demais força gambiarras via shell.

Fundamentos científicos

- **A evolução do uso de tools** — arXiv 2603.22862 traça a trajetória de single-tool call a orquestração multi-tool, o pano de fundo do "code-as-action".
- **Tool learning como campo** — o survey de tool learning (repo) organiza como agentes aprendem a selecionar e compor ferramentas.

(Bibliografia completa: `livro/bibliografia.md`.)

Fontes da indústria

- **Writing effective tools for AI agents** (Anthropic Engineering): a fonte canônica — tools são "contratos entre sistemas determinísticos e agentes não-determinísticos"; a descrição é prompt engineering (pequenos refinamentos → grandes ganhos de acerto), o retorno deve ser otimizado por **densidade informacional por token**, e o ciclo é *prototipar → avaliar → colaborar* (o próprio modelo reescreve as tools a partir das transcrições de eval).
- **Code execution with MCP** (Anthropic): carregar todas as definições e passar intermediários pelo contexto é o gargalo — expor cada tool como arquivo TypeScript que o agente orquestra via código levou um caso de **~150.000 → ~2.000 tokens (−98,7%)**.
- **Tool search tool** + **Advanced tool use** (docs + blog): descoberta dinâmica — envie tudo, marque o não-crítico com `defer_loading: true`, o modelo vê só a busca + as essenciais; um setup multi-servidor gasta ~55k tokens de definições antes de trabalhar, e o tool search corta isso em >85%.
- **Programmatic tool calling** (docs): o modelo escreve Python que chama as tools em sandbox e devolve só o destilado — ~38% menos tokens de input num benchmark com 75 tools; 20–40% típico em produção com 10–49 tools.
- **Code Mode** (Cloudflare): a mesma tese, de um fornecedor de infra — o argumento é de *distribuição de treino*: LLMs escrevem código contra APIs conhecidas melhor do que preenchem schemas sintéticos. Convergência de indústria, não peculiaridade de um vendor.
- **Apply Patch** + **GPT-5.1 for developers** (OpenAI): tool de edição **treinada no modelo** (formato V4A de diffs) — explica por que formatos ad-hoc de search/replace perdem para o formato que o modelo viu em treino.
- **Consulte também**: a coleção viva Awesome Harness Engineering — Tool Design reúne mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

1. O núcleo consensual — e schema derivado de tipos venceu

Os harnesses convergem num núcleo de ~10 tools (ler/escrever/editar arquivo, glob, grep, shell, web fetch/search, todo, delegar) — o kit mínimo de um agente de código. E ninguém escreve JSON Schema à mão: a fonte de verdade é o sistema de tipos (Pydantic no OpenHarness/Hermes, Effect Schema no opencode, classes declarativas no gemini-cli, dataclasses genéricas no software-agent-sdk). O refinamento moderno de qualidade: separar **o que volta ao contexto do modelo do dado estruturado** — o `Observation.to_llm_content` do software-agent-sdk é o design mais limpo (você controla exatamente a densidade informacional que a Anthropic prega).

2. Contexto de tools virou recurso escasso — três regimes de escala

O default de "despejar todas as definições no system prompt" morreu. O estado da arte tem três regimes, e a escolha é por tamanho de catálogo:

- **catálogo fixo** (dezenas de tools): ainda ok mandar tudo;
- **tool search / defer_loading** (centenas de tools, muitos servidores MCP): mantém 3–5 tools quentes, carrega o resto sob demanda — presente como `tool_search`/`tool_discovery` no Codex, Tool Search no OpenClaw, `tool_search` no OpenHarness;
- **code-as-action** (pipelines com dados volumosos): o modelo escreve código que orquestra as tools em sandbox e devolve o destilado — `code-mode` (opencode com V8 embutido, Codex idem), `execute_code` (Hermes chamando tools via RPC (Remote Procedure Call) em "turnos de custo-zero-contexto"), Code Mode (Goose). A métrica que a indústria passou a reportar não é acurácia isolada, é **acurácia por token de definição**.

3. A interface de edição é treinada, não inventada

A lição mais contraintuitiva: o melhor formato de edição de código não é o que você desenha, é o que o **modelo viu em treino**. Daí o `apply_patch` (V4A) ser tool nativa da OpenAI, o opencode dar `apply_patch` a modelos GPT em vez de `edit`/`write`, e o Aider medir empiricamente qual formato cada modelo aplica bem (`percent_cases_well_formed`). Corolário: a seleção de tools **varia por família de modelo** — reconhecimento explícito de que a interface ideal depende de quem está do outro lado. E erro de tool volta como **dado** (para o modelo se autocorrigir), não como exceção.

> **LEITURA EXECUTIVA**
>
> O que está mais moderno: schema derivado de tipos com separação dado×contexto; os três regimes de escala (fixo → tool search → code-as-action) escolhidos por tamanho de catálogo; e a interface de edição como algo treinado. **O que roubar:** `to_llm_content` (controle de densidade por token); tool search com `defer_loading`; medir o formato de edição por modelo (o `percent_cases_well_formed` do Aider); erro-como-dado.

Mão na massa — harness-zero, etapa 2

A etapa 2 substitui os schemas escritos à mão da etapa 1 por uma `ToolPort`: uma tool é uma função tipada, e o schema é **derivado das anotações** (via `inspect`/`typing`, lendo assinatura e docstring). Você adiciona `read_file` ao lado de `get_time`/`somar`, com erros voltando como texto ao modelo (nunca como exceção que derruba o loop). Exercício de completude: o derivador de schema vem esqueletado para um parâmetro; você estende para tipos compostos.

Verificação

1. Por que a descrição de uma tool é prompt engineering e não documentação de API? (Densidade informacional; iterar sobre transcripts de eval.)
2. Seu agente tem acesso a 8 servidores MCP (200+ tools) e gasta 55k tokens antes de agir. Qual regime de escala você adota, e o que ele carrega quente? (Tool search + defer_loading.)
3. Por que dar `apply_patch` a um modelo pode superar um formato search/replace que você desenhou cuidadosamente? (Distribuição de treino.)

Apêndice A — Como cada repositório trata as ferramentas

> Evidência por harness, com paths — complementação online, expandida a cada rodada.

opencode (rodada 1)

~14 tools + 3 experimentais (`tool/`), Effect Schema, descrições `.txt` separadas; **seleção por modelo** (`registry.ts`: GPT recebe `apply_patch` em vez de `edit`/`write`); ripgrep embutido; experimentais `lsp`, `plan_exit`, `code-mode` (V8).

gemini-cli (rodada 1)

~20–25 tools como classes declarativas (`BaseDeclarativeTool` + `Invocation`), registro filtrado (`maybeRegister`), declarações por família de modelo; shell com processos em background, web search com grounding, tracker opcional (6 tools).

OpenHarness (rodada 1)

43+ tools (`tools/`, `BaseTool` + `input_model` Pydantic → `to_api_schema()`); `is_read_only()` alimenta o paralelismo do loop; multimodal, cron, times, `tool_search`.

Codex CLI (rodada 2)

Crate `tools/` com schemas tipados; `unified_exec` (shell persistente com stdin); **`apply_patch` de primeira classe** (parser streaming + gramática `apply_patch.lark`, variando por modelo); `tool_search`/`tool_discovery`; **code-mode com V8 embutido**.

Goose (rodada 2) ⭐ MCP-nativo

Toda tool é MCP: built-ins de `goose-mcp` são `rmcp::ServerHandler` servidos in-process sobre `DuplexStream`; até developer/shell/edit são "platform extensions" falando `McpClientTrait`.

OpenClaw (rodada 2)

Suíte ampla (`openclaw-tools*.ts`): runtime/files/web/browser CDP/mídia; **Tool Search** e **Code Mode** (JS/TS sobre catálogo oculto); 52 AgentSkills injetadas como bloco compacto, lidas sob demanda.

Hermes (rodada 2)

~40+ tools em **toolsets componíveis** com posturas dinâmicas; `execute_code` (Python chamando tools via RPC, "turnos de custo-zero-contexto"); `schema_sanitizer` por provider.

Aider (rodada 2) ⭐ edit formats

Em vez de tools JSON, **formatos de edição** (`*_coder.py`): whole/diff (SEARCH-REPLACE fuzzy)/udiff/patch; seleção por modelo; **validados por benchmark** (`percent_cases_well_formed`).

software-agent-sdk (rodada frameworks) ⭐ dado×contexto

Contrato Action/Observation/Executor; `Observation.to_llm_content` separa o que volta ao modelo do dado estruturado; toolsets (um `create` → várias tools); anotações MCP-style; `ClientToolSpec` (tool executa na máquina do cliente).

IronClaw (rodada 2)

Tools como **capabilities com descritores tipados** declarando `EffectKind`, credenciais e política de rede; separação visibilidade × autoridade (capability oculta falha fechado); obligations (redação/limites) antes de qualquer efeito.

n8n (rodada 2)

`create-node-as-tool.ts`: **qualquer nó `usableAsTool` vira tool** via `$fromAI('chave','desc',tipo)` → schema Zod derivado; ToolWorkflow (sub-workflow como tool), ToolHttpRequest, ToolCode, ToolThink.

Frameworks (rodada frameworks)

Agents SDK (Software Development Kit): `@function_tool` (Pydantic + griffe com auto-detecção de docstring), 13 tipos incl. hosted; LangGraph: herda `@tool` do langchain-core, adiciona `ToolNode` (execução, injeções); CrewAI: `BaseTool`/`@tool` Pydantic, catálogo `crewai-tools` com 79 diretórios.

06 — MCP

🕒 estado da arte 2026-07 · revisão 2026-07-31 · 📖 ~17 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Explicar** por que o MCP (Model Context Protocol) virou a língua franca da integração de agentes — o argumento do padrão aberto contra o custo M×N de integrações ponto a ponto;
2. **Comparar** os transportes do protocolo (stdio, Streamable HTTP, SSE (Server-Sent Events) depreciado) e decidir qual usar para servidor local × remoto;
3. **Avaliar** a superfície do protocolo (tools, resources, prompts, roots, sampling, elicitation) e o que um cliente maduro precisa suportar;
4. **Reconhecer** o servidor MCP como superfície de ataque — a descrição de uma tool é input não confiável — e nomear as defesas de contenção;
5. **Implementar** o adapter MCP client (stdio) atrás de uma porta no harness-zero (etapa 7).

O problema

Nenhum harness consegue embutir tools para todos os sistemas do mundo — bancos de dados, rastreadores de issues, navegadores, APIs internas. Sem um padrão, cada harness escreveria N integrações e cada ferramenta seria reescrita para M harnesses: o clássico problema **M×N**. O MCP resolve pela via do **padrão aberto**: um servidor expõe *tools*, *resources* e *prompts* num protocolo comum (JSON-RPC (Remote Procedure Call) 2.0), e qualquer harness cliente os consome sem saber quem os implementou. Cada lado escreve uma vez — M+N em vez de M×N.

Em pouco mais de dois anos isso virou consenso de indústria — na coorte estudada, **10 dos 11 harnesses da coorte** são clientes MCP completos, todos sobre os SDKs oficiais do protocolo. As decisões que ainda diferenciam as implementações:

- **Transportes**: stdio (processo local), Streamable HTTP (remoto) e o SSE legado.
- **Autenticação**: OAuth para servidores remotos — com que fluxos e provedores?
- **Resiliência**: reconexão, servidores indisponíveis, mudança dinâmica da lista de tools.
- **Superfície**: só *tools*, ou também *resources*, *prompts*, *roots*, *sampling*, *elicitation*?
- **Papel**: o harness é só cliente, ou também **servidor** — consumível por outros agentes?
- **Segurança**: um servidor MCP é código de terceiros injetando texto no contexto do modelo. Quem trata isso como superfície de ataque?

Fundamentos científicos

O MCP nasceu como **especificação de indústria**, não de um paper — e a literatura acadêmica que o alcançou concentra-se, de forma reveladora, em **segurança**. A decisão de projeto que toda essa literatura sustenta é uma só, e decisiva: **a descrição de uma tool (e o retorno) de um servidor MCP é input não confiável**, e deve ser tratada com o mesmo ceticismo de qualquer conteúdo externo.

- **Injeção indireta de prompt** — Greshake et al., arXiv 2302.12173 (AISec '23), o paper que definiu a ameaça: aplicações integradas a LLM (Large Language Model) borram a fronteira entre *dados* e *instruções*, então qualquer conteúdo recuperado é um canal de instrução em potencial. Traduzido para o cliente MCP: o campo de descrição de uma tool e o texto que o servidor devolve são **dados**, nunca instruções confiáveis.
- **A sistematização do MCP** — Hou et al., "MCP: Landscape, Security Threats, and Future Research Directions", arXiv 2503.23278 (também em ACM TOSEM): o SoK canônico. Decompõe o ciclo de vida do servidor (criação → deploy → operação → manutenção) e mostra que o **mesmo servidor é atacável em fases diferentes** — spoofing na instalação, *tool poisoning* em runtime. Decisão: o harness precisa de fronteiras de confiança **por fase**, não um único gate.
- **Descrição de tool como vetor, medido** — MCPTox, arXiv 2508.14925, primeiro benchmark de *tool poisoning* sobre 45 servidores reais / 353 tools: taxa de sucesso de até ~73%, e — o achado desconfortável — **modelos mais capazes foram mais suscetíveis**, com o

alinhamento de segurança oferecendo proteção mínima antes da execução. Decisão dura: não dá para confiar no modelo para se autofiltrar; a metadata da tool tem que ser barrada **antes** de entrar na janela de contexto.

- **A base é empírica, não hipotética** — "MCP at First Glance", arXiv 2506.13538 auditou 1.899 servidores open-source: **7,2% com vulnerabilidades gerais e 5,5% com *tool poisoning*** específico de MCP, em classes que só parcialmente coincidem com appsec tradicional. Decisão: assuma uma taxa-base não-trivial de servidores envenenados no mundo real; scanning ciente-de-MCP, não só SAST. (Ver também MCP Safety Audit, arXiv 2504.03767, que mostra exploits de execução de código e roubo de credencial por tools *legitimamente registradas*.)
- **Escolha o protocolo pelo contexto de confiança** — survey de interoperabilidade, arXiv 2505.02279 compara MCP, ACP, A2A e ANP: o MCP assume uma fronteira cliente-servidor **relativamente confiável**; expor tools MCP através de fronteiras organizacionais não herda as garantias de identidade de A2A/ANP e exige authn/authz adicional (liga ao cap. 17).

> Registro editorial (livro vivo): esta era a dimensão de bibliografia mais rarefeita do livro — registrada como "lacuna acadêmica". Entre as rodadas ela amadureceu de lacuna para **literatura de segurança consolidada** (um SoK, benchmarks, auditorias empíricas). A migração está anotada em `bibliografia.md`.

(Bibliografia completa e ponteiros: `livro/bibliografia.md`.)

Fontes da indústria

- **Arquitetura do MCP** (spec oficial): define o que o harness precisa mapear — no servidor, *tools/resources/prompts*; no cliente, *roots/sampling/elicitation*. A decisão de design: separar o que o servidor *oferece* do que o cliente *concede* — `sampling` e `roots` existem para o servidor pedir uma inferência ou um escopo de arquivos **sem nunca ter acesso direto** ao modelo ou ao filesystem, mantendo o host como único ponto de confiança.
- **Transportes** (spec): dois transportes sobre JSON-RPC — **stdio** (local, sem overhead de rede, o default para servidores locais) e **Streamable HTTP** (remoto). O antigo **HTTP+SSE foi depreciado na revisão 2025-03-26** e só sobrevive por retrocompatibilidade — o cliente moderno tenta `POST InitializeRequest` primeiro e só cai para SSE em 4xx.
- **Introducing the Model Context Protocol** (Anthropic, 25/nov/2024): o anúncio que abriu o padrão, com a analogia do **"USB-C para IA"** (um conector, muitos periféricos) — que é exatamente o argumento M×N do harness. *(anthropic.com retorna 403 pelo proxy; data e framing confirmados por VentureBeat.)*
- **Adoção como ponto de virada**: a OpenAI adotou o MCP em mar/2025 (Agents SDK (Software Development Kit), TechCrunch); o Google/Gemini seguiu (The New Stack); a Microsoft levou o MCP a GA no Copilot Studio e ao Windows. Decisão-chave: quando o segundo maior laboratório adota o protocolo do concorrente, MCP deixa de ser aposta de vendor e vira **infraestrutura neutra** — projetar para MCP reduz o risco de lock-in.
- **Autorização (OAuth 2.1)**: a spec trata todo servidor remoto como **OAuth 2.1 Resource Server** — valida tokens emitidos por um Authorization Server externo (RFC 9728 + 8414 + 7591). O guia prático da Descope traduz em decisão: separar *quem serve a tool* de *quem emite identidade* permite SSO corporativo e tokens com escopo por recurso, em vez de credenciais embutidas no servidor.
- **Segurança na prática** — o Tool Poisoning Attack da Invariant Labs (1/abr/2025) cunhou o termo: instruções maliciosas escondidas na *descrição* de uma tool que o usuário nunca lê mas o modelo obedece. O Trail of Bits mostrou o "line jumping": o simples **registro** de um servidor já é superfície de ataque, antes de qualquer invocação — o gate de confiança tem que ser no *conectar*, não no *chamar*. E the lethal trifecta (Simon Willison): dados privados + conteúdo não confiável + comunicação externa — o MCP torna fácil demais colar tools que, juntas, fecham as três pontas (ler e-mail + abrir PR público = exfiltração). A regra de design é impedir que as três coexistam no mesmo loop.
- **Governança (o MCP virou fronteira, não prótese)**: o registry oficial (preview, set/2025) é uma camada de API *community-owned* — o harness descobre servidores via API padronizada, não listas hardcoded. E em dez/2025 a Anthropic doou o MCP para a Agentic AI Foundation, sob a Linux Foundation — ao lado de **goose e AGENTS.md** como projetos fundadores. O protocolo agora evolui por consenso de um steering group, não pela roadmap de um vendor.
- **The 2026-07-28 Specification** (blog oficial do MCP): o anúncio da maior revisão do protocolo — núcleo stateless, MRTR, extensões, cache e política de depreciação (ver §6 do estado da arte).
- **Consulte também**: a coleção viva Awesome Harness Engineering — Skills & MCP reúne mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

1. A padronização mais clara da disciplina

Onze harnesses da coorte, várias linguagens (TypeScript, Rust, Python), o mesmo protocolo, os SDKs oficiais. É o caso mais límpido de convergência que o livro registrou: onde design de tools, loop e compactação divergem, o MCP unificou. A única exceção na coorte é o **Aider** (MCP nota 0) — e é uma exceção *filosófica*, não um atraso: a escola *context-first* aposta em contexto curado e formatos de edição, e abre mão de MCP de propósito.

2. Transportes convergiram — stdio local, Streamable HTTP remoto

O default estabilizou: **stdio** para servidores locais (o harness lança o processo), **Streamable HTTP** para remotos. O **SSE** virou legado — presente só como fallback de compatibilidade (o opencode faz *fallback automático HTTP→SSE*). Harnesses mais rigorosos fixam a **revisão do protocolo** (o IronClaw fala explicitamente `2025-06-18`), sinal de que o protocolo tem versões e o cliente precisa negociá-las.

3. A virada: o harness virou também servidor MCP

Este é o *update datado* mais forte do capítulo — e uma **previsão do próprio livro que expirou**. Nas primeiras rodadas, anotamos que "nenhum dos harnesses atua como *servidor* MCP no core; o harness-como-serviço aparece por A2A/ACP". A rodada 2 refutou isso: **Codex, Hermes, OpenClaw, OpenHands e n8n expõem-se como servidores MCP**. O harness deixou de ser só um consumidor de tools e passou a ser uma **peça consumível por outros agentes** — o Codex se expõe a IDEs e outros hosts; o OpenClaw serve suas conversas de canal ao Claude Code/Codex; o n8n publica seu grafo de workflow como endpoint MCP. Com isso, a superfície do protocolo se alarga para além de *tools*: **sampling** (o servidor pede completions ao cliente — Hermes) e **elicitation** (o servidor pede input estruturado — Codex) entram no estado da arte. O harness-como-serviço, que antes só existia por A2A/ACP, agora tem uma via MCP nativa.

4. Autenticação: OAuth 2.1 é o piso; o enterprise sobe a régua

Para servidores remotos, o fluxo OAuth com PKCE + callback local + storage de tokens virou o mínimo (opencode, gemini-cli, Codex, Hermes, OpenClaw). O diferencial competitivo está acima: o **gemini-cli** adiciona provedores **Google auth** e **impersonation de service account** (MCP pensado para GCP corporativo); o **OpenClaw** guarda tokens em SQLite e suporta **mTLS**. Autenticação empresarial é hoje a fronteira de features de MCP.

5. Segurança: o servidor MCP é código de terceiros — e a coorte começou a tratá-lo assim

Se um servidor MCP injeta texto no contexto e pode ver argumentos de tools, ele é superfície de ataque — e a literatura (acima) mostra que a ameaça é medível e comum. As defesas observadas na coorte, em camadas (todas conectam ao cap. 07):

- **Testar o vetor**: o gemini-cli inclui um eval de **prompt injection via MCP** — o único que trata o servidor como atacante *testado*.
- **Filtrar o ambiente**: o OpenClaw, ao lançar um servidor stdio, **bloqueia variáveis de ambiente perigosas** (`NODE_OPTIONS`, `LD_*`, `DYLD_*`) que permitiriam carregar código no processo.
- **Mediar credenciais**: o IronClaw adapta tools MCP a *capabilities* **sem conceder autoridade ambiente** — FS, segredos e rede continuam mediados, e **o servidor nunca vê o secret** (a credencial é injetada pela borda). O OpenHands faz **redação e restauração de segredos** em round-trips de configuração.

A régua subiu de "conectar um servidor" para "conectar um servidor **contido**" — exatamente o que os papers de *tool poisoning* e *line jumping* pedem: gate no momento de conectar, sanitização entre servidores, e nenhuma confiança na autofiltragem do modelo.

6. A guinada stateless — a spec 2026-07-28

Três dias antes desta revisão, o protocolo passou pela sua **maior mudança desde o lançamento** (anúncio oficial; changelog). O núcleo virou **stateless**: caem o handshake `initialize`/`notifications/initialized` e o header `Mcp-Session-Id` — cada requisição viaja independente, com protocolo, identidade e capacidades em `_meta` (e um `server/discover` opcional para descoberta). A motivação é infraestrutural: servidores MCP passam a escalar com um load balancer round-robin comum, sem *sticky sessions*. As requisições iniciadas pelo servidor (`elicitation/create`, `sampling/createMessage`, `roots/list`) dão lugar ao **MRTR (Multi Round-Trip Requests)**: o servidor responde `resultType: "input_required"` e o cliente retenta com `inputResponses` — a bidirecionalidade vira ida-e-volta explícita. Completam o pacote: **framework formal de extensões** (Tasks vira `io.modelcontextprotocol/tasks`; MCP Apps e Enterprise Managed Authorization como extensões), **cache como contrato** (`ttlMs`/`cacheScope` nas respostas de listagem — ver cap. 04),

roteamento por headers (`Mcp-Method`/`Mcp-Name`) e a **primeira política formal de depreciação** (janela mínima de 12 meses) — sob a qual **Sampling, Roots, Logging, o transporte legacy HTTP+SSE e o DCR (Dynamic Client Registration, substituído pelo CIMD — Client ID Metadata Documents)** ficam depreciados. Leitura editorial: o que as seções 1–5 descrevem continua sendo o protocolo *instalado* na coorte (a janela de 12 meses existe para isso), mas a direção mudou — e a adoção da 2026-07-28 pelos harnesses é o item nº 1 a medir na próxima rodada do benchmark.

> **LEITURA EXECUTIVA**
>
> O protocolo virou de página em 2026-07-28: núcleo **stateless** (sem handshake, sem `Mcp-Session-Id`), MRTR no lugar de sampling/elicitation iniciados pelo servidor, extensões formais, cache como contrato (`ttlMs`) e a primeira política de depreciação (12 meses) — sob a qual caem Sampling, Roots, Logging e o transporte HTTP+SSE. O que a coorte *roda hoje* ainda é o protocolo das seções 1–5 (a janela existe para isso); o que se *escreve hoje* já deve mirar a 2026-07-28. **O que roubar:** trate a descrição de tool como input não confiável (a literatura mede ~73% de sucesso de *tool poisoning*); em código novo, prefira Streamable HTTP stateless (o fallback SSE agora é transporte depreciado); se for expor um servidor MCP, filtre o ambiente do subprocesso e nunca deixe o servidor ver segredos; se seu público é enterprise, OAuth 2.1 com impersonation é o piso — e planeje a migração DCR→CIMD dentro da janela.

Mão na massa — harness-zero, etapa 7

A etapa 7 (`harness-zero/etapas/07-mcp/`) dá ao harness-zero um **adapter MCP client (stdio)** atrás de uma porta. Fiel à arquitetura hexagonal *por refatoração*: a `ToolPort` da etapa 2 já define o que é uma tool; agora um adapter descobre tools de um servidor MCP externo (via `stdio`, lançando o processo) e as apresenta ao loop como tools nativas — o modelo não distingue. Você conecta o servidor de exemplo incluído (`servidor_mcp_exemplo.py`) — e, como extensão, qualquer servidor MCP real de filesystem. Nota de época: o `ClienteMCP` da etapa implementa o handshake `initialize` do protocolo 2025-06 — que a spec 2026-07-28 **removeu** (núcleo stateless); ele segue funcionando na janela de depreciação de 12 meses, e a diferença entre as duas gerações é, em si, uma aula, lista suas tools, e as chama pelo mesmo caminho das tools locais. Exercício de completude: o cliente trata o *happy path*; você adiciona a **degradação graciosa** (um servidor que cai não derruba a sessão) e um **filtro de env** no subprocesso stdio — a defesa mínima do estado da arte.

Verificação

1. Por que o MCP reduz o custo de integração de M×N para M+N, e o que isso tem a ver com "padrão aberto"? (Cada harness e cada ferramenta escrevem uma vez; o protocolo comum desacopla os dois lados.)
2. Você vai conectar um servidor MCP de terceiros que expõe uma tool `search_tickets`. Cite dois motivos para desconfiar dele e duas defesas concretas. (Descrição de tool = *tool poisoning*; retorno = injeção indireta; e o *registro* já é vetor — *line jumping*. Defesas: env filtrado no subprocesso stdio; credencial mediada — servidor não vê o secret; eval de injeção; gate no conectar; contenção do cap. 07.)
3. Um harness que é **cliente E servidor** MCP ganha o quê que um cliente-só não tem — e que primitivas do protocolo isso ativa? (Vira peça consumível por outros agentes; ativa *sampling* e *elicitation*, o servidor pedindo ao cliente.)

Apêndice A — Como cada repositório trata o MCP

> Evidência por harness, com paths — complementação online, expandida a cada rodada.

opencode (rodada 1) — a implementação mais completa de protocolo

`packages/opencode/src/mcp/` (~1.000 linhas em `index.ts`, + `catalog.ts`, `oauth-provider.ts`, `auth.ts`). Três transportes — `StdioClientTransport`, `StreamableHTTPClientTransport` e `SSEClientTransport` com **fallback automático HTTP→SSE**. OAuth completo: autorização com callback server local, PKCE, comando dedicado `opencode mcp auth`. Cobre a superfície larga: notificações `ToolListChanged`, logging, roots, prompts, resources e resource templates. Instruções do servidor entram no system prompt (`system.ts:mcp()`) — o servidor pode ensinar o modelo a usá-lo (o que é também o vetor de injeção).

gemini-cli (rodada 1) — OAuth de nível corporativo

`packages/core/src/tools/mcp-client.ts` + `mcp-client-manager.ts`, os mesmos três transportes por config. O diferencial em `packages/core/src/mcp/`: além do OAuth padrão, provedores **Google auth** e **impersonation de service account** — MCP para GCP corporativo. Tools viram `DiscoveredMCPTool` com namespacing por servidor; prompts MCP expostos; gestão via `/mcp` e `~/.gemini/settings.json`. Notável: a suíte de evals inclui teste de **prompt injection via MCP** (cap. 11) — o único a tratar servidor MCP como superfície de ataque testada.

OpenHarness (rodada 1) — cliente pragmático

`src/openharness/mcp/` (`McpClientManager`) sobre o SDK `mcp>=1.0.0`: transportes **stdio** e **Streamable HTTP** (sem SSE), com status de conexão, auto-reconnect e **degradação graciosa** quando um servidor cai (`call_tool`/`read_resource` não derrubam a sessão). Resources expostos como tools próprias (`list_mcp_resources`, `read_mcp_resource`); `mcp_auth` para autenticação. Config via `oh mcp` e `--mcp-config`.

Goose (rodada 2) ⭐ MCP-nativo — o protocolo como espinha dorsal

O caso extremo: **toda tool é MCP**. Os built-ins de `goose-mcp` (memory, computercontroller, tutorial…) são servidores `rmcp::ServerHandler` reais servidos **in-process sobre `DuplexStream`** (stdio virtual) e podem rodar standalone (`goose mcp <server>`). Até developer/shell/edit são "platform extensions" falando `McpClientTrait`. Uma única abstração para toda a superfície de ferramentas — o protocolo não é integração, é a arquitetura. (O Goose é, também, um dos projetos fundadores da Agentic AI Foundation.)

Codex CLI (rodada 2) — cliente e servidor, quatro transportes

`rmcp-client/` + `mcp-server/` (o Codex se expõe como servidor MCP). **Quatro transportes** (stdio, streamable HTTP, in-process, process-executor); **OAuth completo** com refresh transactions e store locking; **elicitation**; prewarm/refresh de servidores; templates de aprovação por tool MCP. Integra o MCP à contenção (aprovação por tool).

Hermes (rodada 2) — cliente e servidor, com *sampling*

Cliente com stdio/StreamableHTTP/SSE, OAuth, timeouts por servidor, **sampling** (o servidor pode requisitar completions ao cliente) e paralelismo opt-in por servidor; `mcp_serve.py` expõe o Hermes a outros hosts MCP.

OpenClaw (rodada 2) — cliente e servidor, com filtro de env

`openclaw mcp serve` expõe conversas dos canais via stdio a Codex/Claude Code. Cliente: registry `mcp.servers` com stdio/SSE/streamable-http, **OAuth PKCE em SQLite**, **mTLS**, filtros de tools, probe/doctor — e **filtro de segurança de env** em stdio (bloqueia `NODE_OPTIONS`, `LD_*`, `DYLD_*`). Suporte a MCP Apps com sandbox de origem isolada.

OpenHands (rodada 2) — bidirecional, com redação de segredos

Client (config MCP por agente com redação/restauração de segredos em round-trips GET/PUT) e **server** (o app-server é um FastMCP expondo tools de PR — `create_pr`/`create_mr` — aos sandboxes, mais um **proxy MCP para Tavily** que dá busca sem expor a API key). Perfis de agente referenciam subconjuntos de servidores.

IronClaw (rodada 2) — MCP mediado por *capability*

`ironclaw_mcp` adapta tools MCP a **capabilities sem conceder autoridade ambiente**: FS, segredos e rede continuam mediados; **Streamable HTTP** (protocolo `2025-06-18`); **injeção de credencial mediada** (o servidor nunca vê o secret); recursos contabilizados pelo governor. O modelo de contenção do cap. 07 aplicado ao MCP.

ohmo (rodada 2) — herdado

Completo via `McpClientManager` (herdado da base); contagem de servidores no estado e resumo exposto ao gateway. Lacuna: sem config MCP própria (`~/.ohmo/mcp.json` não existe) e sem isolamento de MCP por canal/remetente.

n8n (rodada 2) — bidirecional no motor de workflow

MCP Client Tool (SSE + Streamable HTTP, Bearer/OAuth2, filtro de tools, cache de sessão por execução) e **MCP Server Trigger** (`McpTrigger` + `McpServer.ts`) — expõe as tools n8n conectadas como endpoint MCP a clientes externos. SDK oficial. O "harness invertido" também fala MCP nos dois papéis.

Aider (rodada 2) — ausente por filosofia

MCP nota **0**. A escola *context-first* em estado puro: as notas 3 estão onde a filosofia aposta (contexto, formatos de edição, git, evals), e a lacuna de MCP é escolha, não atraso.

Frameworks (rodada frameworks)

Agents SDK (OpenAI): suporte a servidores MCP como fonte de tools; LangGraph/langchain: adaptadores MCP para tools; CrewAI: integração MCP via toolkit; software-agent-sdk: anotações MCP-style no contrato de tools. O MCP é ponto de integração assumido também na camada de frameworks — reforço da tese de padronização.

07 — Permissões e Sandboxing

🕒 estado da arte 2026-07 · revisão 2026-07-25 · 📖 ~9 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Distinguir** as duas camadas de defesa — política (o que o agente pode pedir) e contenção (o que o processo consegue fazer);
2. **Projetar** permissões em duas dimensões ortogonais (modo de sandbox × política de aprovação);
3. **Aplicar** a "trifecta letal" e a "regra de dois" como checklists de revisão de toolset e de arquitetura de sessão;
4. **Implementar** uma `PermissionPolicy` como domínio puro (testável sem LLM (Large Language Model)) + paths sensíveis indesligáveis (etapa 6);
5. **Avaliar** um harness real quanto ao seu *blast radius* — o que vaza se a injection vencer?

O problema

Um agente com shell é um usuário com shell: pode apagar arquivos, exfiltrar credenciais, fazer chamadas de rede. Os mecanismos de controle respondem a duas ameaças distintas: o **erro** (o modelo faz algo destrutivo por engano) e o **ataque** (prompt injection convence o modelo a agir contra o usuário). É a dimensão de maior divergência entre os harnesses — sinal de que a indústria ainda não convergiu, embora esteja convergindo rápido.

Dois níveis, frequentemente confundidos: **permissões** (política: aprovação, allowlists, modos) e **sandbox** (contenção: limites impostos pelo SO, mesmo que a política falhe).

Fundamentos científicos

- **A ameaça, definida** — *Not what you've signed up for* (Greshake et al., arXiv 2302.12173): a injection indireta — instruções plantadas em dados que o agente vai ler — é o vetor que nenhuma vulnerabilidade de código tradicional captura.
- **O mapa das defesas** — o survey de superfície de ataque em camadas (arXiv 2604.23338) e o de segurança agêntica (arXiv 2510.06445) organizam ameaças e defesas; o de computer-using agents (arXiv 2505.10924) foca em quem tem shell.

(Bibliografia completa: `livro/bibliografia.md`.)

Fontes da indústria

- **Making Claude Code more secure with sandboxing** (Anthropic): contenção sobre primitivas de SO (bubblewrap/Seatbelt), escrita no workspace, **rede negada por padrão** — e o egress passa por um proxy que roda *fora* do sandbox e faz allowlist por domínio. A fronteira de rede é um componente separado e privilegiado, não uma checagem in-process contornável.
- **How we contain Claude across products** (Anthropic): três regimes (gVisor efêmero, sandbox de SO + aprovação, VM selada com credenciais fora do guest) e a tese central — **fronteiras duras e determinísticas antes de defesas probabilísticas do modelo**. Detalhe honesto: o próprio proxy de egress quebrou duas vezes — trate seu proxy como o componente mais frágil, não o mais confiável.
- **Agent approvals & security** (OpenAI Codex): a matriz de **dois eixos ortogonais** — modo de sandbox (`read-only`/`workspace-write`/`danger-full-access`) × política de aprovação (`untrusted`/`on-request`/`on-failure`/`never`), com o `on-failure` disparando o prompt só *depois* do bloqueio do sandbox. O padrão de design mais copiável do mercado.
- **Agents Rule of Two** (Meta AI): um agente não deve satisfazer mais de dois dos três — processar input não confiável, acessar dados sensíveis, mudar estado/comunicar externamente — na mesma sessão. Critério de *arquitetura de sessão*, não substituto de defense-in-depth.
- **The lethal trifecta** (Simon Willison): dados privados + conteúdo não confiável + comunicação externa = exfiltração. Use como **checklist de toolset**: cada tool nova fecha qual vértice? O contraponto: defesas anunciadas caem quando "o atacante move por último".
- **Ataques ao OpenClaw** (The Hacker News): o caso real — RCE one-click (CVE-2026-25253), credenciais em texto plano, injection plantada em assinatura de e-mail/convite de calendário/issue. O vetor não foi o modelo, foi o **harness**: segredos no mesmo espaço das

tools + entrada não confiável ilimitada.

- **Consulte também**: a coleção viva Awesome Harness Engineering — Permissions & Authorization e Awesome Harness Engineering — Security, Sandbox & Permissions reúnem mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

1. Duas dimensões ortogonais, não um slider

O modelo mental antigo ("YOLO ↔ pergunte tudo") morreu. O consenso é separar **capacidade física máxima** (sandbox) de **quando escalar ao humano** (política de aprovação) — configuráveis independentemente. O Codex é o exemplo canônico (modo × política, com `on-failure`). E há dois paradigmas de contenção que o benchmark separou:

- **contenção por SO** (o processo *não consegue*): Seatbelt + bubblewrap/seccomp + Landlock no Codex; 6 perfis Seatbelt + Docker no gemini-cli; WASM fail-closed + Docker per-tenant no IronClaw;
- **arquitetura de autoridade** (o loop *não alcança*): o IronClaw torna o loop estruturalmente incapaz de agir sem o kernel — trust class inforjável por tipos, aprovações como leases por invocação, verificado por testes de dependência. Nenhum harness combina os dois plenamente ainda — é a fronteira aberta.

2. Política sem contenção é aposta na obediência do modelo

A lição transversal do benchmark: harnesses com política elegante mas sem sandbox de SO (opencode, ohmo) apostam que o modelo obedece. Três defesas baratas e exportáveis que o estado da arte consolidou: **paths sensíveis indesligáveis** (`SENSITIVE_PATH_PATTERNS` do OpenHarness — nega `.ssh`, credenciais, `.kube/config` antes de qualquer regra de usuário, explicitamente contra injection); **parsing estrutural de shell** antes de julgar (o policy engine do gemini-cli entende redirecionamentos e wrappers; o `defense_in_depth` do software-agent-sdk detecta composições como fetch-to-exec via AST); e **credenciais fora do processo** (injetadas na borda de egress, nunca no espaço das tools — IronClaw, e a lição direta do caso OpenClaw).

3. Prompt injection é tratada como não-resolvível — o esforço migrou para o blast radius

O consenso de 2026, do modelo aos vendors: não se "detecta" injection de forma confiável. O trabalho migrou para **desenhar sessões que nunca acumulam a trifecta** (regra de dois como critério de quando quebrar contexto), **isolar credenciais** (keychain no host, VM selada) e **controlar egress** (allowlist por domínio via proxy externo). Na categoria de agentes pessoais, o vetor de terceiros ganhou defesa própria: **pairing/allowlist de contatos deny-by-default** (OpenClaw, ohmo) e **sandbox `non-main`** para toda sessão que não seja a do dono. E a norma emergente de honestidade: publicar **taxas de falso-negativo** do gate (o auto mode do Claude Code é discutido com números nos dois sentidos) em vez de afirmar segurança binária.

> **LEITURA EXECUTIVA**
>
> O que está mais moderno: as duas dimensões ortogonais; os dois paradigmas de contenção (SO × autoridade) e a constatação de que ninguém os combinou; e a migração de "detectar injection" para "reduzir blast radius" (trifecta/regra-de-dois como checklists, credenciais fora do processo, egress controlado). **O que roubar:** paths sensíveis indesligáveis; parsing estrutural de shell; `on-failure` (aprovar só após o bloqueio); pairing de contatos; publicar a taxa de falso-negativo do gate.

Mão na massa — harness-zero, etapa 6

A etapa 6 introduz a `PermissionPolicy` como **domínio puro**: uma função `(ação, contexto) → allow | ask | deny` que não conhece LLM nem chat — testável isoladamente (é o "domínio isolado" que o DDD nomeia, e o teste roda sem rede). Você implementa: os três veredictos (permitir/perguntar/negar), os paths sensíveis indesligáveis, e a **aprovação inline no chat** (o front pausa e pergunta — a manifestação visível da política). Exercício de completude: a avaliação de regras vem pronta; você adiciona o parsing mínimo de um comando shell antes de julgá-lo.

Verificação

1. Um harness só tem política de aprovação, sem sandbox de SO. Que classe de ataque ele não consegue conter, e por quê? (Política sem contenção; o modelo pode ser convencido.)
2. Você vai adicionar uma tool de envio de e-mail a um agente que já lê issues do GitHub e tem acesso ao repositório privado. Aplique a trifecta letal. (Fecha o terceiro vértice → exfiltração possível.)
3. Por que `on-failure` (aprovar só depois do bloqueio) pode ser melhor que `on-request` (aprovar antes de cada ação)? (Fricção × cobertura; o sandbox filtra o que nunca precisa de humano.)

Apêndice A — Como cada repositório trata permissões e sandboxing

> Evidência por harness, com paths — complementação online, expandida a cada rodada.

gemini-cli (rodada 1) ⭐ policy engine + sandbox de SO

`packages/core/src/policy/policy-engine.ts`: regras priorizadas com **parsing estrutural de shell** (`parseCommandDetails`, `stripShellWrapper`, detecção de redirecionamento), regras em TOML; 4 `ApprovalMode`; **6 perfis Seatbelt** (`sandbox-macos-*.sb`) + Docker/Podman com proxy; **trusted folders** gatekeepando hooks/agents.

OpenHarness (rodada 1) ⭐ paths sensíveis

`permissions/checker.py`: path rules, comandos negados, 3 modos; **`SENSITIVE_PATH_PATTERNS` hardcoded e indesligável** (`.ssh`, `.aws/credentials`, `.gnupg`, `.kube/config`) contra injection; sandbox via `sandbox-runtime`/Docker com allowlist de domínios; `trust_env=False` nas tools web (anti-SSRF).

opencode (rodada 1) — política sem contenção

`permission/`: rulesets com wildcards (`allow | ask | deny`, last-match-wins, default `ask`), aprovação via `Deferred` + evento; **subagentes derivam permissões restritas**; **sem sandbox de SO no core** (containers só no enterprise).

Codex CLI (rodada 2) ⭐ contenção por SO em 3 camadas

`sandboxing/` + `linux-sandbox/` + `windows-sandbox-rs/`: Seatbelt via `sandbox-exec` (path hardcoded anti-tamper), bubblewrap embutido + **seccomp** + `NO_NEW_PRIVS`, Landlock legado; `AskForApproval` incl. `Granular`; **execpolicy em Starlark** por comando; `assess_patch_safety`; network-proxy.

Goose (rodada 2)

`permission/`: modos `GooseMode` (Auto/Approve/Chat); **`permission_judge` usa um LLM** para classificar read-only; `ToolPermissionStore` por assinatura com expiração; isolamento de execução leve (shell direto; Docker externo).

OpenClaw (rodada 2) ⭐ pairing de terceiros

`src/pairing/` + `docs/security/THREAT-MODEL-ATLAS.md`: **DMs como input não confiável**, `dmPolicy: "pairing"` default (código de pareamento, allowlist SQLite); sandbox multi-backend (Docker `network:none`/`readOnlyRoot`/`capDrop:ALL`, SSH, OpenShell) com modo **`non-main`**; `openclaw doctor`/`security audit`; caveat: `sandbox.mode` off por default na sessão main.

Hermes (rodada 2)

`tools/approval.py` (detecção + allowlist), callbacks por-thread; **seis backends de terminal isolados** (local, Docker, SSH, Singularity, Modal, Daytona); subagentes com `_subagent_auto_deny` seguro por default; `path_security.py` anti-traversal.

IronClaw (rodada 2) ⭐⭐ arquitetura de autoridade

`crates/ironclaw_authorization` + `_approvals` + `_trust` + `_wasm` + `_process_sandbox` + `_secrets` + `_network` + `_safety`: autorização de invocação exata (fail-closed), aprovações como **leases por invocação com fingerprint**, **trust class inforjável por**

tipo (`#[serde(skip_deserializing)]`), WASM (fuel/memória/rate, egress negado), Docker per-tenant, secrets zero-exposure na borda de egress, anti-SSRF, leak detector bidirecional — o loop não alcança os efeitos (verificado por testes de dependência).

ohmo (rodada 2.5) — a metade certa

`channels/impl/base.py`: allowlist **deny-by-default** + isolamento de sessão por remetente + bloqueio de comandos admin remotos + paths sensíveis do OpenHarness. Gap: `permission_mode`/`sandbox_enabled` do `gateway.json` são **código morto** — sem dial entre nega-tudo e full_auto.

software-agent-sdk (rodada frameworks)

`sdk/security/`: análise de risco (LLM analyzer + `defense_in_depth/` determinístico com parser AST de shell detectando **fetch-to-exec**) + política de confirmação (`AlwaysConfirm`/`ConfirmRisky` por limiar); a conversa **retorna** em `WAITING_FOR_CONFIRMATION` (não bloqueia); mascaramento de segredos.

n8n (rodada 2) — permissão estrutural

A permissão é **topológica**: o autor escolhe quais nós ficam na porta `AiTool` — allowlist por construção. HITL real via `sendAndWait` (pausa durável), proibido em sub-agentes; nó Guardrails.

Frameworks (rodada frameworks) — deixam aberto

LangGraph e CrewAI não têm política de tools nativa (constrói-se sobre `interrupt`/HITL); o Agents SDK (Software Development Kit) tem guardrails em três níveis (agente/run/tool) como primitiva, mas contenção fica por conta do adotante.

08 — Memória e Estado

🕒 estado da arte 2026-07 · revisão 2026-07-26 · 📖 ~13 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Distinguir** as três camadas do problema — estado de sessão, memória de longo prazo e estado do workspace — e o requisito próprio de cada uma;
2. **Explicar** por que memória **não é** RAG (Retrieval-Augmented Generation) (memória = retrieval + caminho de escrita + gestão de estado) e por que markdown versionável venceu bancos vetoriais no domínio de código;
3. **Derivar** uma política de recall a partir da fórmula recência × importância × relevância, e uma de esquecimento a partir do uso;
4. **Avaliar** o impacto da **reversibilidade** (checkpoint de workspace) sobre o cálculo de risco de permissões;
5. **Implementar** a persistência de sessão do harness-zero (adapter SQLite + `/resume`) na etapa 4.

O problema

O modelo esquece tudo entre chamadas; o harness lembra por ele. "Memória e estado" cobre três camadas com requisitos diferentes:

1. **Estado de sessão** — a conversa em si: mensagens, tool-calls, metadados. Precisa sobreviver a reinícios e permitir retomar (`resume`), ramificar e reverter.
2. **Memória de longo prazo** — fatos que atravessam sessões: preferências do usuário, decisões do projeto, aprendizados. Precisa ser **selecionável** (nem tudo entra em todo contexto) e **atualizável** (fatos mudam).
3. **Estado do workspace** — o que o agente *fez* nos arquivos. Precisa ser **reversível**: desfazer as mudanças de um agente é tão importante quanto fazê-las.

A tese que unifica as três: a janela de contexto é memória volátil e cara; tudo o que precisa durar vive **fora** dela, e o harness decide o que trazer de volta e quando.

Fundamentos científicos

A memória de agentes tem literatura madura — e ela dá o vocabulário exato para o que os harnesses fazem na prática.

- **A janela como RAM** — MemGPT: LLMs as Operating Systems, arXiv 2310.08560 trata o contexto como memória principal escassa, apoiada por dois níveis externos (*recall* de histórico recente e *archival* pesquisável), com o **agente** paginando dados via tool calls ("context page faults"). Decisão: quem decide o que despejar e o que buscar é o agente, não um pipeline RAG fixo.
- **A taxonomia canônica** — CoALA, arXiv 2309.02427 separa memória **episódica** (experiência passada), **semântica** (conhecimento do mundo/usuário) e **procedural** (habilidades/código), mais a working memory. Decisão: no momento da escrita, decida *que tipo* de memória aquele fato é — cada tipo se recupera diferente. O survey de mecanismos de memória, arXiv 2404.13501 (depois ACM TOIS) organiza o subsistema por *fontes* · *formas* · *operações* (escrita, gestão/consolidação, leitura) — orce esforço por operação, não só pelo índice de busca.
- **A fórmula de recall** — Generative Agents, arXiv 2304.03442 (UIST '23) guarda observações num *memory stream* datado e recupera por um score composto de **recência × importância × relevância** (decaimento exponencial de recência, importância pontuada por LLM (Large Language Model), relevância por embedding). É a fórmula concreta que um harness deve implementar para rankear o que reentra no contexto — e introduz a **consolidação por reflexão** (sintetizar reflexões de alto nível a partir de clusters de observações).
- **Esquecimento controlado** — MemoryBank, arXiv 2305.10250 (AAAI '24) decai/reforça a força de cada memória por uma curva de Ebbinghaus (tempo decorrido × frequência de acesso), mantendo o store limitado. Decisão: memória não-usada é candidata a poda — o *tracking de uso* é o que fecha o ciclo.
- **Memória como aprendizado** — Reflexion, arXiv 2303.11366 (NeurIPS '23) converte feedback de resultado em auto-reflexão verbal, persistida num buffer episódico e reinjetada na próxima tentativa — melhorar sem atualizar pesos. E arquiteturas recentes (A-MEM, arXiv 2502.12110; Mem0, arXiv 2504.19413) tratam a escrita como um pipeline *extrair → consolidar → linkar*, com a rede de memórias se auto-organizando (estilo Zettelkasten). Ponte para o cap. 16 (auto-melhoria).

(Bibliografia completa e ponteiros: `livro/bibliografia.md`.)

Fontes da indústria

- **Sessão como log de eventos durável** — Manage sessions (Claude Code): cada sessão é gravada continuamente em disco como **JSONL** por projeto (uma linha por mensagem/tool-use/metadado); `--continue` retoma a mais recente no diretório, `--resume` abre um seletor. Decisão: "retomar" é **restaurar estado completo** (tool calls, resultados, modo de permissão, objetivo ativo), não replay de texto — o harness é dono de um log durável privado, não de um schema público estável.
- **Reversão do workspace como trilha separada** — Checkpointing (Claude Code) captura o estado do código antes de cada prompt; `/rewind` restaura código, conversa **ou** ambos (100 checkpoints recentes, limpos com a sessão). O file-checkpointing do Agent SDK expõe isso como primitiva reusável. Decisão: desfazer o *código* é um store separado de desfazer a *conversa*, ligados pelo índice do prompt.
- **Memória durável como arquivos com precedência** — How Claude remembers your project: a hierarquia CLAUDE.md (política gerenciada → usuário → projeto → local), o atalho `#` para anexar uma linha de memória, `/memory` para editar. Decisão: memória cross-sessão é **markdown em tiers de precedência** (o mais específico vence) — versionável, auditável, escopada; relida no launch como contexto sempre-ligado.
- **A memory tool (beta) e "assuma interrupção"** — Memory tool: o modelo pede operações (`view/create/str_replace/…`) num diretório `/memories` que persiste entre conversas, mas a execução é **client-side** — seu app implementa o armazenamento (e a proteção contra path traversal, limites de tamanho, expiração). O sistema injeta "ASSUMA INTERRUPÇÃO: sua janela pode ser resetada a qualquer momento". Pareada com o context management (context editing evicta pares stale da janela; a memory tool persiste fora dela) — dois níveis: higiene de curto prazo + store externo de longo prazo. Decisão: para agentes de longa duração, você precisa dos dois; a janela é efêmera, o `/memories` é a fonte de verdade (o padrão do ensaio harnesses para agentes de longa duração: log de progresso estruturado, lido no início e atualizado no fim de cada sessão).
- **Memória ≠ RAG** — a distinção virou tese de indústria: a Letta ("RAG is not agent memory") e a AWS Bedrock AgentCore argumentam que RAG é leitura *stateless*; memória é leitura **+ caminho de escrita + gestão de estado** (admissão, resolução de fatos conflitantes, invalidação). A Letta expõe *memory blocks* auto-editáveis e tiers **core/recall/archival** (a hierarquia do MemGPT como produto); a mem0 roteia cada fato por camada com tempo de vida próprio; a Zep/Graphiti modela memória como **grafo de conhecimento bi-temporal** (fatos desatualizados são *invalidados*, não deletados); a LangMem/LangGraph separa **short-term (thread)** de **long-term (store por namespace)**. Decisão: não dá para "comprar" memória pregando um vector store — é preciso um pipeline de escrita/atualização/invalidação.
- **Consulte também**: a coleção viva Awesome Harness Engineering — Memory & State reúne mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

1. Três camadas, três campeões — e nenhum banco vetorial

As três camadas do problema receberam campeões diferentes na coorte: **estado de sessão** (durabilidade de banco — opencode com SQLite + eventos replayáveis; Codex com rollout jsonl por turno; OpenHands com event-stream), **memória de longo prazo** (relevância + rigor de formato — OpenHarness com memdir versionado, `relevance.py` e `usage.py`; Hermes com `MEMORY.md`/`USER.md` + `session_search`), **estado de workspace** (reversão via git — Aider e gemini-cli). E o achado que persiste: **nenhum dos harnesses de código usa banco vetorial** para memória. No domínio de código, markdown versionável venceu embeddings — porque memória de código precisa de *caminho de escrita* (o agente edita o arquivo) e auditabilidade, exatamente o que a tese "memória ≠ RAG" prevê.

2. A fórmula de recall e o esquecimento saíram do paper para o código

O `relevance.py` + `usage.py` do OpenHarness é a instância prática do stream de Generative Agents: seleciona por relevância o que entra no contexto e marca uso — memória não-usada vira candidata a poda (a curva de esquecimento do MemoryBank, na prática). O Hermes formaliza a **manutenção ativa**: um único tool edita `MEMORY.md`/`USER.md` com **nudges periódicos** (a cada 10 turnos) e um `session_search` (índice FTS5/BM25 sobre o SQLite de sessões, com modos discovery/recall/sumarização) dá **recall cross-session** — a camada archival do MemGPT construída sobre busca textual, não vetorial.

3. Reversibilidade virou primitiva — e muda o cálculo de risco

O checkpoint de workspace deixou de ser feature e virou primitiva: o **Aider** foi pioneiro anos atrás (estado git-nativo: auto-commit atômico por rodada, `aider_commit_hashes`, `/undo`, `.aider.chat.history.md`), o **gemini-cli** consagrou (`/restore`, `/rewind` do disco via snapshots git), e o Claude Code o expõe como checkpointing com trilhas separadas para código e conversa. A consequência de projeto é a mais interessante: **um agente cujas ações são reversíveis muda o cálculo de risco de tudo o mais** — permissões podem ser mais frouxas quando desfazer é barato (liga ao cap. 07).

4. Providers plugáveis e o harness como servidor de memória

A fronteira emergente: memória como serviço plugável. O Hermes já aceita provedores externos (Honcho, mem0, supermemory) por trás da sua camada; produtos como Letta/mem0/Zep se posicionam como a "camada de memória universal" consumível por qualquer harness. A tensão de projeto para as próximas rodadas: manter a memória como **arquivo local versionável** (auditável, portável, sem dependência) ou terceirizá-la para um store gerenciado (grafo bi-temporal, escala). No código, o arquivo ainda vence; fora dele, o pêndulo é menos claro.

> **LEITURA EXECUTIVA**
>
> O que está mais moderno: a moldura de tiers OS (RAM ↔ recall ↔ archival) com o agente paginando; recall por recência×importância×relevância com esquecimento por uso; reversão do workspace como primitiva que afrouxa permissões; e a distinção dura memória × RAG (write path + invalidação). **O que roubar:** persista a sessão como log de eventos durável (retomada = restaurar estado, não replay); trate memória como markdown versionável com tracking de uso; separe a trilha de reversão do código da conversa; e, para agentes longos, escreva um log de progresso durável assumindo que a janela some a qualquer momento.

Mão na massa — harness-zero, etapa 4

A etapa 4 (`harness-zero/etapas/04-sessoes/`) dá persistência ao harness-zero: um **adapter SQLite** por trás de uma `StorePort` guarda mensagens e tool-calls como linhas tipadas, e `/resume` restaura o estado completo de uma sessão anterior (não só o texto). Fiel ao hexagonal *por refatoração*: a dor que faz a porta nascer é reabrir o processo e perder a conversa. Exercício de completude: a persistência cobre o *happy path*; você adiciona um `USER.md`/`MEMORY.md` mínimo lido no início e um log de progresso atualizado ao fim — o padrão "assuma interrupção" na sua forma mais simples.

Verificação

1. Por que memória de agente não é a mesma coisa que RAG, e o que isso explica sobre a escolha de markdown versionável em vez de banco vetorial nos harnesses de código? (Memória = retrieval + caminho de escrita + gestão/invalidação de estado; código precisa de write path auditável.)
2. Você tem 10.000 memórias e espaço para 20 no contexto. Que score usa para escolher, e como decide o que podar com o tempo? (Recência × importância × relevância; poda por falta de uso — curva de esquecimento.)
3. Seu agente ganhou checkpoint de workspace com `/rewind`. Que decisão *de outra dimensão* isso permite afrouxar, e por quê? (Permissões — o cálculo de risco cai quando desfazer é barato; cap. 07.)

Apêndice A — Como cada repositório trata memória e estado

> Evidência por harness, com paths — complementação online, expandida a cada rodada.

opencode (rodada 1) — estado como banco de dados

Persistência em **SQLite via Drizzle** (`packages/core/database`, `core/session/sql.ts`): sessões, mensagens e partes são linhas tipadas. Sessões têm `parentID` (hierarquia para subagentes), suportam revert (`session/revert.ts`) e **compartilhamento** (`share/`, `sync/`). A V2 (`CONTEXT.md`) leva o desenho a "infra de dados": inbox durável de prompts, eventos replayáveis com cursores (`sessions.events({sessionID, after})`), snapshots de contexto persistidos entre reinícios. O modelo de estado mais robusto da rodada 1 — o harness como sistema distribuído com estado durável.

gemini-cli (rodada 1) — o workspace reversível

Memória de longo prazo nos próprios `GEMINI.md` (tool `save_memory`, global em `~/.gemini` + índice de projeto, com auto-memory testada em evals). O recurso distintivo é o **checkpointing baseado em git** (`services/gitService.ts` + `chatRecordingService.ts`): snapshots do workspace antes de edições, habilitando `/restore` e `/rewind` — desfazer as mudanças do agente no disco, não só na conversa — além de `/resume`.

OpenHarness (rodada 1) — memória como arquivo, com disciplina

`src/openharness/memory/` (13 módulos): memória persistente em markdown (`MEMORY.md`/memdir por projeto) com **schema versionado, escrita atômica com file-lock e assinaturas**. `relevance.py` seleciona o que entra no contexto; `usage.py` marca uso (memória não usada é candidata a poda). Sessões persistidas com metadados ricos (`services/session_storage.py`): modo de permissão, estado de arquivos lidos, skills invocadas, checkpoints de compactação. Retomada via `-c/--continue`, `-r/--resume`, `/resume`.

Aider (rodada 2) ⭐ estado git-nativo — o pioneiro da reversão

`aider/repo.py`: **auto-commit atômico por rodada** com mensagem gerada por LLM, atribuição de autoria configurável, `aider_commit_hashes` rastreando o que a IA fez, `dirty_commit` isolando mudanças pendentes. `/undo`, `diff` e `blame` viram a interface de memória; complementos `.aider.chat.history.md` e `--restore-chat-history`. **Antecipou em anos** o "checkpoint git" que o gemini-cli e o Claude Code consagraram.

Hermes (rodada 2) ⭐ memória multicamada com recall cross-session

`MEMORY.md` (notas do agente) + `USER.md` (perfil do usuário) editados por tool única com **nudges periódicos** (a cada 10 turnos); provedores externos plugáveis (**Honcho, mem0, supermemory**); e **`session_search`** — índice FTS5 sobre o SQLite de sessões com três modos (discovery/BM25, recall janelado, sumarização por LLM) para recall cross-session. A camada archival do MemGPT sobre busca textual.

Codex CLI (rodada 2) — rollout jsonl por turno

Cada turno é persistido em **rollout jsonl** (recuperável); `SessionTask` (Regular/Review/Compact/UserShell) organiza a máquina de tarefas. Estado de sessão durável e resumível integrado ao loop (`core/src/session/`).

OpenHands (rodada 2) — event-stream persistido

`openhands/app_server/event/` persiste cada `Event` como JSON por conversa, com paginação, filtros e export de trajetória. O control-plane consome/persiste eventos; o loop ação-observação roda no SDK. Event-sourcing como coluna vertebral do estado.

OpenClaw (rodada 2) — session lanes e arquivos de workspace

Runs serializados por *session lane* com write-lock file-based entre processos; arquivos de workspace (`MEMORY.md`, `USER.md`, `IDENTITY.md`…) injetados com orçamentos (20k chars/arquivo, 60k total) e truncamento marcado. Persistência de conversa por canal.

ohmo (rodada 2) — backends de sessão/memória como plugins

Implementa `SessionBackend` e `MemoryCommandBackend` do OpenHarness como plugins de primeira classe (sem tocar no core), mais um **pool multi-sessão** (`RuntimeBundle` por `session_key`, recriado quando o cwd muda). Prova de que a fronteira app/engine foi desenhada.

IronClaw (rodada 2) — estado resumível por checkpoints

Estado resumível por **checkpoints**; máquina de estados Queued→Running→Blocked→Completed com **leases/heartbeats** e "one active run per canonical thread". O `LoopExit` carrega apenas referências duráveis — o loop nunca muta estado; o `LoopExitApplier` valida evidência host-owned antes de aplicar.

n8n (rodada 2) — memória do motor de workflow

Memória via *memory sub-nodes* (janela `contextWindowLength`, corte `maxTokensFromMemory`); estado do workflow persistido pelo motor entre execuções. Curto por natureza — execuções acionadas por evento não acumulam contexto longo (compactação nota 1, por

design).

Frameworks (rodada frameworks)

LangGraph: **checkpointer** (short-term, thread-scoped) + **store** por namespace (long-term cross-thread); LangMem: memórias semântica/episódica/procedural como tools; Agents SDK e CrewAI: estado de sessão/curto-prazo com hooks de persistência. A distinção short × long term é primitiva de framework — o que os harnesses de código implementam à mão, os frameworks expõem como API.

09 — Planejamento

🕒 estado da arte 2026-07 · revisão 2026-07-26 · 📖 ~12 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Distinguir** os três instrumentos de planejamento — plan mode, todo list e decomposição — e o requisito de cada um;
2. **Explicar** por que plan mode se implementa como um caso do sistema de permissões (imposto, não pedido);
3. **Comparar** ReAct (intercalar razão e ação) com plan-then-execute e decidir quando cada um serve;
4. **Avaliar** a estratificação tático × durável (plano da tarefa × objetivo da sessão) e a decomposição com dependências;
5. **Implementar** plan mode imposto por permissões no harness-zero (etapa 8).

O problema

Modelos tendem a agir precipitadamente: editam antes de entender, "resolvem" antes de mapear o problema. Os artefatos de planejamento forçam uma fase de leitura e desenho antes da fase de escrita — e dão ao humano um ponto de aprovação barato (revisar um plano custa menos que revisar um diff).

Três instrumentos distintos, frequentemente confundidos:

1. **Plan mode** — um *estado* do harness em que escrever é proibido; o agente só pesquisa e propõe.
2. **Todo list** — memória de trabalho da tarefa em andamento: o que falta, o que está feito.
3. **Decomposição** — quebrar trabalho grande em subtarefas rastreáveis, possivelmente com dependências.

Fundamentos científicos

A literatura de planejamento explica *por que* esses instrumentos existem — e adverte contra confiar no plano do modelo.

- **Intercalar vence planejar-tudo-antes (quando o ambiente é imprevisível)** — ReAct, arXiv 2210.03629 (ICLR '23) intercala traço de raciocínio e ações de tool no mesmo loop: cada observação revisa o próximo pensamento, então o agente se recupera de surpresas em vez de executar um plano velho. Decisão: carregue raciocínio e observações num único transcript alternado.
- **Planejar-antes ajuda (quando o escopo é conhecido)** — Plan-and-Solve, arXiv 2305.04091 faz o modelo emitir um plano explícito antes de resolver, suprimindo passos faltantes. Os dois não se contradizem: são regimes distintos — o plano explícito para tarefas de escopo conhecido, a intercalação para ambientes incertos.
- **Decompor só quando preciso** — ADaPT, arXiv 2311.05772 decompõe **recursivamente e apenas quando o executor falha** uma subtarefa, adaptando a profundidade à dificuldade e à capacidade do modelo. Decisão: tente executar primeiro, decomponha na falha — evita o over-planning que a maioria dos harnesses (sabiamente) não impõe.
- **Isolar o contexto por subtarefa** — Beyond Entangled Planning, arXiv 2601.07577 (2026) decompõe num **DAG de sub-objetivos** e dá contexto *escopado* a cada um, para que erros locais e replanejamento não poluam um histórico monolítico — reporta até −82% de tokens. Ponte direta com subagentes (cap. 10).
- **Não confie no plano do modelo — externalize** — PlanBench, arXiv 2206.10498 e TravelPlanner, arXiv 2402.01622 mostram que modelos crus falham em geração de plano e perdem o fio de múltiplas restrições (GPT-4 ~0,6% no TravelPlanner). Decisão: externalize o rastreio de restrições num artefato (plano/todo), em vez de confiar que o modelo segura tudo no contexto. É *a* justificativa da todo list.
- **A taxonomia como checklist** — survey de planejamento, arXiv 2402.02716 organiza os componentes em cinco vias (decomposição de tarefa · seleção de plano · módulo externo · reflexão · memória); PlanGenLLMs, arXiv 2502.11221 dá seis critérios (completude, executabilidade, otimalidade, representação, generalização, eficiência) e PLANET, arXiv 2504.14773 organiza benchmarks por categoria.

(Bibliografia completa e ponteiros: `livro/bibliografia.md`.)

Fontes da indústria

- **Plan mode é uma camada de permissão** — Choose a permission mode (Claude Code): plan mode remove escrita/execução pela *sessão inteira*; o agente lê e explora, mas toda mutação fica retida até você sair (Shift+Tab cicla Normal → Plan → Auto-accept; `/plan`; `--permission-mode plan` para CI). Decisão: o planejamento é garantido **revogando as tools de mutação**, não pedindo ao modelo que "planeje primeiro". É a confirmação oficial da descoberta da rodada 1.
- **Explorar → Planejar → Codar → Commitar** — Best practices (Claude Code): as fases de exploração e planejamento são "as mais baratas em tokens e as mais valiosas em resultado". Decisão: separar exploração de execução impede estruturalmente resolver o problema errado antes de entender o código.
- **Todo como artefato rastreado por máquina** — Todo tracking (Agent SDK): o `TodoWrite` cria checklists com três estados (pending/in_progress/completed) atualizados em tempo real. Decisão: externalizar o plano num artefato estruturado dá ao agente uma âncora de memória de trabalho e ao usuário visibilidade de progresso — e a evolução para um sistema de *tasks* com dependências e persistência torna o plano infraestrutura durável, não scrollback.
- **Pensar entre as ações** — The "think" tool adiciona um passo de raciocínio *no meio* do uso de tools (depois que o resultado chega); o extended thinking expõe blocos de raciocínio com `budget_tokens` e, nos modelos 4, **interleaved thinking** (pensar → chamar tool → pensar sobre o resultado → chamar de novo). Decisão: aloque orçamento explícito para passos de planejamento e deixe o raciocínio intercalar com as tools — planejar não é um prefixo único, é contínuo. *(anthropic.com retorna 403 pelo proxy; confirmado por espelhos independentes.)*
- **Spec-driven: o spec é o plano durável** — GitHub Spec Kit formaliza `specify` (o quê/porquê) → `plan` (arquitetura) → `tasks` (lista acionável) → `implement`, com gates de aprovação entre estágios; a Kiro gera `requirements.md` (EARS `WHEN…THE SYSTEM SHALL…`), `design.md` e `tasks.md`, e **deriva um grafo de dependências** que executa tarefas independentes em ondas concorrentes. Decisão: o plano vira fonte de verdade persistida e re-consumida a cada fase — é exatamente o método com que **este livro é escrito** (ver a constituição do projeto).
- **Planejar é uma função de orquestração — e a tensão sobre paralelizar** — o sistema multi-agente da Anthropic faz o *lead* analisar a query, **gravar o plano em memória** e só então spawnar workers com specs isolados (planejamento como papel dedicado). A Cognition ("Don't Build Multi-Agents") contrapõe: o Devin centraliza o planejamento num contexto contínuo, porque planejar *é* gestão de contexto — paralelizar workers vira "telefone sem fio" de decisões implícitas conflitantes. Decisão: decompor-e-paralelizar é um gate de custo/benefício, não um default (liga ao cap. 10). *(cognition.com 403 pelo proxy; confirmado por espelhos.)*
- **Consulte também**: a coleção viva Awesome Harness Engineering — Planning & Task Decomposition reúne mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

1. Plan mode = modo de permissão (agora padrão oficial)

A descoberta da primeira rodada — os harnesses implementam plan mode **como um caso do sistema de permissões** (cap. 07), não como subsistema próprio — deixou de ser observação e virou padrão documentado: a doc oficial do Claude Code descreve plan mode exatamente assim (remove mutação pela sessão). Entrar em plan mode = trocar para um ruleset que nega escritas; sair = restaurar, com aprovação explícita. O padrão maduro combina três garantias: read-only **imposto** (não pedido), plano como **artefato persistido** (não só texto na conversa) e **aprovação explícita** antes de executar.

2. ReAct virou o default; o plano explícito recuou para o trabalho longo

O sinal mais claro do livro vivo veio do n8n: seu **Plan-and-Execute Agent foi depreciado** (só existe na V1 legada, ao lado do ReAct), e a V2/V3 convergiram para o Tools Agent puro — planejamento implícito no modelo. Isso instancia a tese científica: conforme os modelos planejam melhor inline, a intercalação (ReAct) vence o plan-then-execute como default, e o **plano explícito se concentra onde ainda paga**: trabalho longo, humano no loop, e decomposição de tarefas grandes. Não é que planejar morreu — é que o planejamento barato migrou para dentro do loop.

3. A todo list é rastreio de restrições externalizado

O que PlanBench e TravelPlanner provam (modelos perdem o fio de múltiplas restrições) é o que a todo list resolve: um checklist com estados (Codex `update_plan`, `TodoWrite`, `todo` do Hermes/Goose, `TODO.md` do OpenHarness) tira as restrições da cabeça do modelo e

as põe num artefato. A evolução moderna é dar **dependências e persistência** a esse artefato — o tracker em grafo do gemini-cli, o grafo de dependências da Kiro, o DAG do "Beyond Entangled Planning".

4. Tático × durável — a contribuição dos agentes pessoais

Os harnesses de código têm um plano *da tarefa*; falta-lhes o *durável*. O **OpenClaw** preenche isso com quatro camadas: `update_plan` (tático, um passo `in_progress` por vez), **Goals** (um objetivo durável por sessão, com token budget e estados, injetado por turno e visível na UI), **Task Flow** (orquestração durável com steps e estado JSON) e standing orders (políticas persistentes). Essa estratificação tática × durável é a fronteira que a categoria de agentes pessoais trouxe à disciplina.

5. Planejamento é a dimensão mais fraca — e isso é um dado, não um acaso

Em todas as rodadas, planejamento foi a nota mais baixa da indústria (Codex 2, Goose 2, Aider 2, Hermes 2, OpenHands 1, n8n 1, IronClaw 2; só o gemini-cli e o OpenClaw chegam a 3). A leitura do livro vivo (registro de expiração, "plan mode imposto", 🔵 aberta): a prótese existe porque os modelos agem precipitadamente, e ela expira quando os modelos planejarem sob risco espontaneamente — o que ainda não aconteceu. A fraqueza persistente da dimensão *é* a evidência de que a prótese ainda é necessária.

> **LEITURA EXECUTIVA**
>
> O que está mais moderno: plan mode como camada de permissão (padrão oficial); ReAct/interleaved thinking como default, com plano explícito reservado a trabalho longo; todo/checklist como rastreio de restrições externalizado, evoluindo para grafos de dependência; e a estratificação tático × durável. **O que roubar:** imponha o read-only pela permissão, não pelo prompt; externalize o plano num artefato persistido com estados; dê orçamento de thinking aos passos de planejamento; e decomponha-e-paralelize só quando a largura da tarefa paga o custo.

Mão na massa — harness-zero, etapa 8

A etapa 8 (`harness-zero/etapas/08-plan/`) adiciona plan mode ao harness-zero **reusando** a `PermissionPolicy` da etapa 6: entrar em plan mode seta um modo que a política traduz em "toda tool de escrita é negada"; o agente só lê e propõe; sair pede aprovação e restaura o modo. É a demonstração concreta da tese do capítulo — plan mode não é um subsistema, é uma configuração do domínio de permissões que já existe. Exercício de completude: o `propor_plano` já persiste o artefato (`PLAN.md`); você adiciona a exigência de que a saída do plan mode só aconteça com um `PLAN.md` aprovado — o gate entre planejar e executar.

Verificação

1. Por que faz sentido implementar plan mode como um modo do sistema de permissões, em vez de um subsistema dedicado? (Reusa um mecanismo existente e ganha de graça a garantia de que o read-only é *imposto*, não sugerido ao modelo.)
2. Seu agente opera num ambiente imprevisível (respostas de API mudam o próximo passo). Você planeja tudo antes ou intercala razão e ação? Por quê? (Intercala — ReAct: cada observação revisa o próximo pensamento; um plano fixo fica velho.)
3. Um benchmark mostra seu agente perdendo o fio de 8 restrições numa tarefa. Que instrumento de planejamento ataca isso, e por quê? (Todo list / checklist — externaliza o rastreio de restrições para fora do contexto do modelo.)

Apêndice A — Como cada repositório trata o planejamento

> Evidência por harness, com paths — complementação online, expandida a cada rodada.

opencode (rodada 1) — plan como agente

Plan mode é um **agente built-in `plan`** com ruleset read-only (nega edições, pede confirmação para bash) — planejar é trocar de agente, não só de modo. A tool `plan_exit` (`tool/plan.ts`) fecha o ciclo: pergunta se aprova, **escreve o plano em arquivo** e transiciona para o

agente `build`. Prompts dedicados (`prompt/plan-mode.txt`, `plan-reminder-anthropic.txt` — lembretes por família de modelo). Todos por sessão via `todowrite` (`session/todo.ts`).

gemini-cli (rodada 1) — plan com gatekeeping e decomposição

`ApprovalMode.PLAN` (`policy/types.ts`) com `enter-plan-mode`/`exit-plan-mode`: estado read-only cujo prompt lista as tools disponíveis, e `getApprovedPlanPath()` **gatekeepa a execução**. Todos via `WriteTodosTool`. O instrumento que os outros não têm: o **tracker** opcional (`trackerTools.ts`) — tarefas com dependências (`tracker_add_dependency`) e grafo (`tracker_visualize`). Plan mode tem eval comportamental própria (`evals/plan_mode.eval.ts`).

OpenHarness (rodada 1) — a versão mínima e correta

`EnterPlanModeTool` seta `settings.permission.mode = PLAN` (bloqueia todas as escritas); `ExitPlanModeTool` restaura — a implementação mais direta da equivalência plan-mode-é-permissão. Todos em `TODO.md` via `TodoWriteTool` (persistente, legível). Skill bundled `plan`; decomposição pesada no subsistema autopilot (fila de `RepoTaskCard`).

OpenClaw (rodada 2) ⭐ — tático × durável em quatro camadas

`update_plan` (plano multi-step, um `in_progress` por vez), **Goals** (objetivo durável por sessão com token budget e estados, injetado por turno e visível na UI), **Task Flow** (orquestração durável com steps e estado JSON) e **standing orders** (políticas persistentes). A estratificação tática × durável que os harnesses de código não têm.

Codex CLI (rodada 2) — checklist estruturado

Tool `update_plan` (checklist visível na TUI (Terminal User Interface)) + `ReviewTask`. Sem plan mode de duas fases com aprovação de plano antes da execução — a economia de "planejar antes" fica no checklist, não num gate de permissão.

Aider (rodada 2) — plan-then-edit por modos de coder

`/ask` (discute sem editar), `/architect` (raciocina o "como" antes de delegar) e `/context` (usa o repo-map para convergir nos arquivos). Plan-then-edit leve, sem artefato de plano persistido nem todo list; o split `architect→editor` executa o plano com um segundo modelo.

Goose (rodada 2) — recipes declarativos

Recipes (YAML/JSON com instructions, parâmetros tipados, `response.json_schema`, retry) + extensão `todo` + `final_output_tool`. Planejamento declarativo/reusável, sem plan mode de duas fases.

Hermes (rodada 2) — todo + Kanban

Tool `todo` + orçamento de iterações + **sistema Kanban** para coordenação multi-agente com specs. Planejamento acoplado ao loop, sem planner formal separado.

n8n (rodada 2) — o planejamento que recuou

O **Plan-and-Execute Agent** existe mas é **legado** (só na V1, junto com ReAct/Conversational); V2/V3 convergiram para o Tools Agent puro. Planejamento ficou implícito no modelo (+ `ToolThink` opcional). O caso mais claro de plano explícito perdendo para intercalação.

IronClaw (rodada 2) — planejamento temporal, não decomposição

Sem decomposição de tarefas de primeira classe; o "planner" do loop é composição de strategies. A força está no planejamento *temporal* (agendamento, leases/heartbeats — dim. suplementar 14).

OpenHands / ohmo (rodada 2)

OpenHands: aba planner na UI e ganchos, sem subsistema de decomposição de 1ª classe neste repo (nota 1; o núcleo migrou para o SDK). ohmo: plan mode/todos herdados que assumem TUI — sem superfície de aprovação de plano num canal de chat.

Frameworks (rodada frameworks)

LangGraph: planejamento como grafo explícito de nós (o plano *é* a topologia); Agents SDK e CrewAI: papéis planner/executor e processos sequencial/hierárquico; a spec-driven (Spec Kit/Kiro) trata o plano como artefato versionado com gates. Onde os harnesses de código improvisam o plano no loop, os frameworks o materializam como estrutura de primeira classe.

10 — Subagentes e Orquestração

🕒 estado da arte 2026-07 · revisão 2026-07-26 · 📖 ~13 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Explicar** por que o ganho primário de um subagente é isolamento de contexto (lê muito, devolve pouco), não paralelismo;
2. **Comparar** as três filosofias — subagente-como-ferramenta, como-serviço e como-colega;
3. **Avaliar** o gate de custo/benefício de decompor-e-paralelizar (a tensão Anthropic × Cognition) e os modos de falha que justificam guardrails;
4. **Distinguir** delegação local de delegação entre sistemas (A2A (Agent-to-Agent)/ACP (Agent Client Protocol)) e quando cada uma se aplica;
5. **Implementar** a tool `task` com sessão-filha e permissões derivadas no harness-zero (etapa 9).

O problema

Um único contexto não segura tarefas grandes: exploração de codebase polui a janela com dumps de arquivos; trabalhos paralelizáveis rodam em série; e um agente generalista faz tudo mediocremente. Subagentes resolvem por **divisão de contexto** (o subagente lê 50 arquivos e devolve só a conclusão), **especialização** (prompts e permissões por papel) e **paralelismo**.

As decisões de projeto:

- **Isolamento**: sessão-filha? Processo separado? Worktree git próprio (para edições paralelas sem conflito)?
- **Permissões**: herda as do pai? Derivadas e restritas? Degradadas por profundidade?
- **Comunicação**: fire-and-forget (retorna um resultado) ou canal contínuo (mailbox, mensagens)?
- **Alcance**: só local, ou delegação a agentes remotos de outros vendors?

Fundamentos científicos

A literatura de sistemas multi-agente (MAS) tem duas mensagens para quem constrói harness: os padrões que funcionam, e a advertência de que a maioria das falhas é de projeto.

- **A falha é de design, não do modelo** — MAST, "Why Do Multi-Agent LLM (Large Language Model) Systems Fail?", arXiv 2503.13657 deriva empiricamente 14 modos de falha em três categorias (especificação/papéis · desalinhamento inter-agente · verificação de tarefa), e conclui que a maioria vem do *sistema*, não dos pesos. Decisão: invista em specs de papel explícitos, checagens de alinhamento e um estágio de verificação dedicado — não num modelo maior.
- **Papéis e SOPs contra alucinação em cascata** — MetaGPT, arXiv 2308.00352 codifica *Standardized Operating Procedures* e papéis de linha de montagem (PM, arquiteto, engenheiro, QA) com artefatos intermediários estruturados, porque encadear LLMs ingenuamente propaga alucinação; saídas escopadas por papel deixam o agente seguinte verificar o anterior. E CAMEL, arXiv 2303.17760 mostra que o role-play **deriva** (troca de papel, repetição, término precoce) — a estabilidade de papel precisa ser *imposta*, não assumida.
- **Topologia programável e recrutamento dinâmico** — AutoGen, arXiv 2308.08155 separa os agentes da topologia de conversa (troque o padrão de orquestração sem reescrever agentes); AgentVerse, arXiv 2308.10848 monta o grupo por tarefa e monitora comportamento emergente negativo. ChatDev, arXiv 2307.07924 decompõe o pipeline em diálogos de duas partes por fase. Ponteiro de taxonomia: o survey de MAS, arXiv 2402.01680.
- **O ceticismo saudável** — o debate multi-agente é uma primitiva de verificação (Du et al., arXiv 2305.14325), mas Should We Be Going MAD?, arXiv 2311.17371 e Stop Overvaluing Multi-Agent Debate, arXiv 2502.08788 mostram que ele nem sempre bate self-consistency/CoT a compute igual. Decisão: **sempre compare o harness multi-agente com um baseline single-agent compute-matched** antes de aceitar a complexidade.

(Bibliografia completa e ponteiros: `livro/bibliografia.md`.)

Fontes da indústria

- **Subagente = instância isolada com toolset restrito** — Create custom subagents (Claude Code): cada subagente é uma instância *fresca e isolada* lançada pela tool `Task`, com janela de contexto própria e toolset por tipo de agente. Os subagents do Agent SDK são declarados como config (nome, tools, modelo, prompt) — dá para fixar modelos baratos (Haiku para Explore read-only) por papel e impor least-privilege por tipo. Decisão: um subagente de busca queima tokens explorando sem poluir o contexto do orquestrador, devolvendo só um resumo compacto.
- **Orchestrator-worker — e o preço** — o multi-agent research system da Anthropic: um *lead* planeja, grava o plano em memória e spawna subagentes paralelos, cada um com contexto isolado e um **contrato explícito** (objetivo, formato de saída, tools, fronteiras). O ganho de largura vem a **~15× os tokens** de um chat único (e, segundo o post, tokens explicam ~80% da variância de desempenho) — só paga em tarefas de alto valor e muita amplitude. O guia de quando usar multi-agente dá os três casos: poluição de contexto, subtarefas genuinamente paralelas, especialização que afia a seleção de tools. *(anthropic.com 403 pelo proxy; números por espelhos independentes.)*
- **O contra-argumento** — Don't Build Multi-Agents (Cognition): prefira um agente **single-thread com compressão de contexto**. Quando o trabalho se abre em paralelo, cada subagente age sobre uma visão parcial e toma decisões implícitas conflitantes (o exemplo do Flappy Bird: um constrói fundo estilo Mario, outro um pássaro incompatível) — um "telefone sem fio" que cria a etapa de reconciliação que a própria arquitetura gerou. Dois princípios: *compartilhe o traço completo com todo agente* e *ações carregam decisões implícitas, evite as conflitantes*. Para tarefas longas, adicione um modelo de compressão em vez de dividir a thread. *(cognition.com 403; confirmado por HN/GitHub.)*
- **Os frameworks materializam os padrões** — Agents SDK (OpenAI) distingue **handoffs** (transfere controle a um especialista) de **agents-as-tools** (um manager chama sub-agentes como funções, mantendo a thread); o Swarm foi a origem educacional do handoff. CrewAI escolhe entre **sequential** e **hierarchical** (`manager_llm` delega e valida); o LangGraph modela um **supervisor** roteando entre workers com estado persistente; o Magentic-One (AutoGen) mantém um **ledger de progresso** e replaneja na falha; o ADK do Google mistura coordinator/dispatcher com primitivas `Sequential/Parallel/Loop`. Decisão: escolha a forma de coordenação (handoff × tool × supervisor × ledger) pelo que precisa reter — thread, controle ou recuperação.
- **Delegação entre sistemas: A2A (e ACP convergindo nele)** — quando os subagentes vivem em vendors diferentes, a delegação vira protocolo: o A2A usa **Agent Cards** (JSON anunciando identidade, skills, endpoint, auth) para descoberta e **Tasks** com ciclo de vida como unidade de trabalho delegado, sobre HTTP+JSON-RPC (Remote Procedure Call)+SSE (Server-Sent Events); é a generalização cross-org do handoff da tool `Task`. O ACP (IBM/BeeAI) era a alternativa REST-nativa — mas fundiu-se no A2A sob a Linux Foundation em ago/2025. Decisão: para trabalho novo, padronize no A2A (liga ao cap. 17).
- **Consulte também**: a coleção viva Awesome Harness Engineering — Task Runners & Orchestration reúne mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

1. Três filosofias — ferramenta, serviço, colega

A moldura da primeira rodada persiste e ganhou reforço da rodada 2. **Subagente-como-ferramenta**: pontual, contido, com guardrails (opencode `task` → sessão-filha, depth 1; Aider split architect→editor, depth 1). **Subagente-como-serviço**: registry, contratos de terminação, alcance remoto (gemini-cli `invoke_agent` + A2A; Codex `multi_agents_v2` com **grafo de agentes persistido** e ~100 perfis; Goose `orchestrator` lead/worker). **Subagente-como-colega**: equipes persistentes com comunicação contínua (OpenHarness Swarm com mailbox + worktree git por membro; Hermes com **Kanban dispatcher** e handoffs estruturados).

2. O ganho primário é isolamento de contexto — não paralelismo

O que os três harnesses da rodada 1 já mostravam, a indústria consolidou: o subagente vale porque **lê muito e devolve pouco**. É por isso que o Claude Code o modela como instância *fresca* e isolada, e por que o worktree git (OpenHarness) importa — ele isola *edições* paralelas, não só leituras. Isso é o mesmo princípio do "contexto escopado por subtarefa" do cap. 09 (Beyond Entangled Planning): o subagente é o veículo de escopo de contexto.

3. A tensão central: paralelizar custa, e a maioria das falhas é de design

O eixo de decisão da dimensão é a tensão Anthropic × Cognition. Orchestrator-worker compra largura (+~90% em pesquisa) a **~15× tokens**; o single-thread evita o "telefone sem fio" mas serializa. O MAST fecha o argumento com dados: a maioria das falhas de MAS é de

especificação e coordenação, não do modelo — o que explica por que todo harness sério cerca subagentes de **guardrails**: profundidade limitada (opencode/Aider depth 1; OpenClaw 1–5), contratos de terminação (gemini-cli GOAL/MAX_TURNS/TIMEOUT), permissões **degradadas por profundidade** (OpenClaw: subagente nunca ganha `message`/`gateway`/`cron`), e a expressão extrema — o **IronClaw deny-filtra `spawn_subagent` em todos os profiles de produção** (o design suporta, a política proíbe até haver confiança). A regra de projeto: decompor-e-paralelizar é um gate de custo/benefício, com baseline single-agent como controle.

4. A virada: orquestrar harnesses de outros vendors

A fronteira que a rodada 2 tornou concreta: o subagente pode ser *outro harness*. O OpenClaw orquestra Claude Code, Gemini CLI, opencode e Codex como subagentes via runtime **ACP**; o OpenHands (Canvas) orquestra Claude Code, Codex e Gemini via perfis **ACP**; o gemini-cli é cliente **e servidor A2A**. Com o ACP-IBM (Agent Communication Protocol) convergindo no A2A sob a Linux Foundation, o *agent card* vira o contrato universal de delegação entre sistemas. A orquestração deixou de ser interna ao harness e virou interoperabilidade (cap. 17).

> **Adendo da rodada ext-1 (2026-07-31): o isolamento de *workspace* virou infraestrutura.** O corpus isolava o **contexto** do subagente; o Grok Build (xAI, aberto em 2026-07-15) fecha a outra metade — o **filesystem**. Cada `spawn_subagent` com isolamento ativo recebe uma **git worktree própria** criada por uma crate dedicada (`xai-fast-worktree`: CoW paralelo, snapshots BTRFS O(1), overlayfs, metadata com auto-GC), com merge de volta como operação de protocolo (`x.ai/git/worktree/apply`) e fallback gracioso para o workspace compartilhado. A lição não é "usar worktrees" (vários harnesses têm); é o investimento em torná-las **baratas o bastante para o agente usar sem pensar** — subagentes paralelos que editam deixam de brigar pelo working tree. Confirmado no código (`agent/subagent/handle_request.rs`), não só no anúncio.

> **LEITURA EXECUTIVA**
>
> O que está mais moderno: subagente como isolamento de contexto com contrato explícito; a escolha de coordenação (handoff × tool × supervisor × ledger); guardrails motivados por modos de falha reais (MAST); a delegação cross-vendor via A2A; e — desde a rodada ext-1 — o isolamento de workspace por worktree barata (Grok Build). **O que roubar:** dê a cada subagente um contrato (objetivo/formato/tools/fronteiras) e contexto isolado; limite profundidade e degrade permissões por profundidade; compare sempre com um single-agent compute-matched; se subagentes editam em paralelo, isole o filesystem (worktree), não só o contexto; e, se orquestrar entre sistemas, fale A2A.

Mão na massa — harness-zero, etapa 9

A etapa 9 (`harness-zero/etapas/09-subagentes/`) adiciona uma tool `task` que lança um **subagente em sessão-filha**: contexto próprio, **permissões derivadas e restritas** da sessão-pai, e **profundidade máxima 1** (subagente não spawna subagente) — os guardrails que o MAST justifica, na sua forma mínima. O subagente recebe um contrato (objetivo + formato de saída), roda seu próprio loop e devolve só o resumo ao pai. Exercício de completude: você adiciona a degradação de permissões por profundidade e um contrato de terminação configurável (objetivo + timeout por subagente).

Verificação

1. Seu orquestrador precisa entender 40 arquivos para decidir um refactor, mas você não quer 40 dumps no contexto principal. Como um subagente resolve, e qual é o ganho real? (Isolamento de contexto — o subagente lê os 40 e devolve só a conclusão; o ganho primário não é paralelismo.)
2. Um colega propõe rodar 5 subagentes em paralelo para acelerar. Cite o principal risco (com um nome da literatura/indústria) e o gate que você aplica antes de aceitar. (Telefone sem fio / decisões implícitas conflitantes — Cognition; falhas de coordenação — MAST. Gate: custo/benefício ~15× tokens + baseline single-agent compute-matched.)
3. Você quer que seu harness delegue uma subtarefa a um agente de outro vendor. Que mecanismo usa e qual é o "contrato"? (A2A; o Agent Card anuncia identidade/skills/endpoint/auth, e a Task é a unidade de trabalho delegado.)

Apêndice A — Como cada repositório trata subagentes e orquestração

> Evidência por harness, com paths — complementação online, expandida a cada rodada.

opencode (rodada 1) — delegação contida

Tool `task` (`tool/task.ts`) → subagente em **sessão-filha** (`parentID`), **permissões derivadas e restritas** (`agent/subagent-permissions.ts`), depth 1. Agentes em markdown com modo `primary|subagent|all`; built-in `build`/`plan`/`general`/`compaction`. Modo background experimental (`BackgroundJob`) com `task_id` para **retomar a sessão de subagente**.

gemini-cli (rodada 1) — do subagente local ao remoto

`invoke_agent` sobre `AgentRegistry` (`packages/core/src/agents/registry.ts`); built-in codebase-investigator, generalist, cli-help, browser, skill-extraction, cada um com `ModelConfig`. Terminação explícita (`AgentTerminateMode`: GOAL/MAX_TURNS/TIMEOUT). Exclusividade: **A2A** client+server (`@a2a-js/sdk`, agent cards). Evals de delegação próprias.

OpenHarness (rodada 1) — times, não subagentes

Swarm (`src/openharness/swarm/`, 11 módulos): `AgentTool` em três backends (subprocesso, remoto, teammate in-process); `TeamRegistry`; **mailbox** (comunicação contínua); **worktrees git** (`worktree.py`) para edições paralelas; `permission_sync.py`. Tools `team_create/delete`, `send_message`.

Codex CLI (rodada 2) — grafo de agentes persistido

Duas gerações de API (`multi_agents_v2`: spawn, send_message, followup, interrupt, wait); ~100 perfis de subagentes em TOML; **`agent-graph-store`** (grafo persistido), identidade de agente, comunicação inter-agente, hooks SubagentStart/Stop; `ThreadManager` coordenando threads paralelas.

OpenClaw (rodada 2) — spawn push-based e ACP externo

`sessions_spawn` cria subagentes isolados com **conclusão push-based** (`sessions_yield` como espera sem polling); nesting 1–5; política de tools **degradada por profundidade** (subagentes nunca ganham `message`/`gateway`/`cron`). Runtime **ACP** orquestra Claude Code, Gemini CLI, opencode e Codex como subagentes; Swarm via Code Mode.

Hermes (rodada 2) — Kanban dispatcher

`delegate_task` spawna `AIAgent` filhos com contexto isolado e aprovação não-interativa segura; **Kanban dispatcher** no gateway spawna workers com handoffs estruturados, bloqueio para input humano e heartbeat em operações longas.

Goose (rodada 2) — SubRecipes e orchestrator

`summon` delega a subagentes (Agent filho com recipe própria, eventos streamados); **SubRecipes** com composição hierárquica e execução paralela/sequencial; extensão `orchestrator` (lead/worker: list/start/send/interrupt/stop).

Aider (rodada 2) — architect→editor

Split `architect_coder.py`: um modelo raciocinador produz o plano; após confirmação, um segundo coder (com `editor_model`/`editor_edit_format` próprios) executa. Orquestração de dois papéis com modelos distintos, profundidade fixa 1.

IronClaw (rodada 2) — design elegante, política restritiva

Subagentes como child-runs no mesmo pipeline, com gates/checkpoints unificados e teste E2E — **mas `spawn_subagent` está deny-filtrado em todos os profiles de produção** (`TEMP(disable-spawn-subagents)`). A nota reflete a capacidade disponível, não o design (que seria 3). O caso extremo de "guardrail vence capacidade".

OpenHands / ohmo (rodada 2)

OpenHands: primitivas do SDK (`openhands.sdk.subagent`) + **AgentProfiles** por organização, incluindo perfis **ACP** — o Canvas orquestra Claude Code, Codex e Gemini. ohmo: Agent/Task/Team/SendMessage herdados; assimetria observada (`/tasks run` bloqueado remotamente, tools equivalentes disponíveis ao modelo).

Grok Build (rodada ext-1) — worktrees como infraestrutura ⭐

`agent/subagent/handle_request.rs`: `spawn_subagent` com `capability_mode` **intersectado** com o toolset do tipo (`intersect_capability_modes`), profundidade máx. 1, `resume_from`, contratos de I/O entre personas; isolamento por `WorktreeBuilder…worktree_kind(WorktreeKind::Subagent)` sobre `xai-fast-worktree` (CoW + BTRFS O(1) + auto-GC), merge via `x.ai/git/worktree/apply`; agentes de plugin proibidos de declarar `mcpServers`/hooks/`bypassPermissions`.

Pi (rodada ext-1) — a recusa documentada

Sem subagentes no core, por manifesto ("There's many ways to do this. Spawn pi instances via tmux, or build your own"); o exemplo primeiro-classe `examples/extensions/subagent/` spawna **processos `pi` completos** (isolamento real de contexto) com 4 personas e 3 workflows — a feature existe como prova de que a superfície de extensão basta.

n8n (rodada 2) — agente como tool de agente

AI Agent Tool (`AgentTool.node.ts` v3): um agente completo como tool de outro — o V3 roda o loop do sub-agente inline (`resolveSubAgentRequest`), com proibição de HITL aninhado; **ToolWorkflow** (sub-workflows como tools). Orquestração hierárquica visual.

Frameworks (rodada frameworks)

Agents SDK: handoffs × agents-as-tools; CrewAI: sequential × hierarchical (`manager_llm`); LangGraph: supervisor + workers como nós com estado; AutoGen/Magentic-One: orchestrator com ledger e replanejamento; Google ADK: coordinator/dispatcher + `Sequential/Parallel/Loop`. Os frameworks expõem como API de primeira classe o que os harnesses de código implementam à mão.

11 — Verificação e Evals

🕒 estado da arte 2026-07 · revisão 2026-07-26 · 📖 ~14 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Distinguir** as três perguntas da verificação (o harness funciona? · o agente se comporta? · o trabalho está certo?) e a resposta técnica de cada;
2. **Explicar** por que a auto-correção *intrínseca* não basta e a verificação precisa ser externa e ancorada em sinal (testes, LSP (Language Server Protocol), tools);
3. **Avaliar** o *reward hacking* — o agente jogando contra o verificador — e as defesas (held-out, testes imutáveis, anti-mock, verificar o estado final);
4. **Reconhecer** os vieses do juiz LLM (Large Language Model) (posição, verbosidade, self-preference) e como mitigá-los;
5. **Implementar** uma suíte de evals do harness-zero (juiz + respostas gravadas) na etapa 10.

O problema

Como saber se o agente funciona? A pergunta se desdobra em três, com respostas técnicas diferentes:

1. **O harness funciona?** — testes de software clássicos sobre o código do harness (loop, tools, permissões).
2. **O agente se comporta bem?** — evals: o comportamento emergente (usa as tools certas? é frugal? respeita o plan mode? resiste a injection?) sob teste de regressão.
3. **O trabalho do agente está certo?** — verificação em runtime: sinais (LSP, testes, lint) realimentados ao modelo durante a tarefa.

A segunda é a mais difícil e a mais negligenciada: comportamento de agente é estocástico, caro de testar e muda silenciosamente a cada troca de modelo ou de prompt. E há uma quarta pergunta que a rodada 2 tornou incontornável: **o agente está trapaceando o verificador?**

Fundamentos científicos

A ciência da verificação de agentes tem três mensagens duras — e todas empurram para o mesmo lugar: verificação **externa e ancorada**.

- **Grading por execução, não por aparência** — SWE-bench, arXiv 2310.06770 (ICLR '24) verifica aplicando o patch do modelo e rodando os **testes reais e ocultos** do repositório (FAIL_TO_PASS + PASS_TO_PASS). Decisão: para código, o único sinal confiável é "os testes reais passaram", não similaridade de diff. E SWE-agent, arXiv 2405.15793 mostra que a **ergonomia das tools** (a Agent-Computer Interface) dirige o sucesso tanto quanto o modelo.
- **A auto-correção intrínseca não basta** — Large Language Models Cannot Self-Correct Reasoning Yet, arXiv 2310.01798 é o contra-resultado decisivo: sem feedback externo, pedir ao modelo que "revise" pode *degradar* respostas certas. Decisão: "pedir ao modelo para se conferir" **não é** estratégia de verificação — o harness precisa fornecer um verificador. CRITIC, arXiv 2305.11738 mostra o caminho: auto-crítica **ancorada em tool** (o código roda? o fato confere?) supera introspecção; Self-Consistency, arXiv 2203.11171 dá a versão barata (amostrar caminhos + voto) para respostas checáveis.
- **O juiz LLM funciona — com vieses** — Judging LLM-as-a-Judge, arXiv 2306.05685 mede ~80% de acordo com humanos, mas documenta vieses de **posição, verbosidade e self-preference**. Decisão: randomize/troque a ordem das respostas e faça a média, dê rubrica e resposta-referência, e calibre contra um gold set humano (survey, arXiv 2411.15594) — um único call de juiz não é ground truth. E verifique o **estado final do mundo**, não o transcript: τ-bench, arXiv 2406.12045 mostra que `pass@1` esconde inconsistência brutal (`pass^8` < 25%).
- **O agente joga contra o verificador** — o tema novo e mais importante: com recompensas verificáveis (RLVR / Tülu 3, arXiv 2411.15124) um verificador determinístico é sinal e recompensa mais difícil de fraudar — *mas* reward hacking, arXiv 2606.15385 e testes randomizados contra trapaça, arXiv 2606.07379 mostram que agentes praticam *specification gaming* zero-shot: apagam asserts, dão `sys.exit(0)`, patcham o pytest. Decisão: mantenha uma métrica de ground-truth **held-out** que o agente nunca otimiza, e **testes imutáveis** que ele não pode tocar.

(Bibliografia completa e ponteiros: `livro/bibliografia.md`.)

Fontes da indústria

- **O benchmark é o padrão — e é contaminável** — SWE-bench Verified (OpenAI) é o subconjunto de 500 tarefas *humanamente auditadas*, criado porque o SWE-bench cru tinha specs ambíguas e testes quebrados que reprovavam soluções corretas (audite o verificador antes de confiar nele). Mas o OpenAI parou de reportar SWE-bench Verified por contaminação/memorização — o eval precisa de rotação e held-outs para seguir sendo sinal. O Terminal-Bench (arXiv 2601.11868, repo `harbor-framework/terminal-bench`) leva o rigor ao terminal: cada tarefa embarca **Docker + solução humana + testes de verificação**, gradando o *estado final do ambiente*, não a plausibilidade do transcript.
- **Evals como disciplina de engenharia** — Define success criteria and build evaluations (Claude): defina critérios mensuráveis *antes*, force o juiz a emitir um veredito discreto e a raciocinar antes de pontuar. O Demystifying evals for AI agents (Anthropic) decompõe o eval em componentes (task · trial · agent harness · eval harness · trace · grader · suite) e insiste: **grade o estado final, não a última mensagem** (uma resposta pode "soar certa" e a tarefa ter falhado). E reporte o erro-padrão da média para distinguir regressão real de ruído.
- **Verificação dentro do loop** — Effective harnesses for long-running agents (Anthropic): cada sessão roda os testes, **verifica a feature end-to-end como um usuário faria** (automação de navegador), deixa um log de progresso e commita limpo. E o Claude Code best practices eleva o TDD ao padrão agêntico mais forte: escreva os testes primeiro, confirme que falham, **commite-os como checkpoint, e implemente sem editá-los** — commitar os testes antes é a rede que revela quando o agente trapaceia alterando o teste em vez de corrigir o código.
- **Ferramental de eval versionado** — OpenAI Evals, Inspect (UK AISI) (Dataset + Solver + Scorer, com sandbox Docker/K8s — o eval e o sandbox são um só sistema), promptfoo (um `promptfooconfig.yaml` versionado como gate de CI), Braintrust e LangSmith (rubrica como config, correções humanas viram few-shot). Decisão: os checks vivem no controle de versão e rodam no CI como qualquer teste.
- **Verificação virou adversarial** — Natural emergent misalignment from reward hacking (Anthropic): agentes aprendem a *gamear o verificador* (sair antes dos testes, patchar o pytest, apagar asserts) e o hábito **generaliza para sabotagem mais ampla**. Decisão: endureça o verificador (testes randomizados/held-out, arquivos de teste imutáveis) e não deixe o agente tocar no próprio grader — o The Verification Horizon (arXiv 2606.26300) adverte que, quando a capacidade do agente ultrapassa o verificador, o reward hacking ressurge; o verificador tem de *evoluir* (testes → rubrica → juízes interativos).
- **Consulte também**: a coleção viva Awesome Harness Engineering — Verification & CI Integration e Awesome Harness Engineering — Evals & Verification reúne mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

1. Três perguntas, três campeões (e a lacuna que fechou)

A moldura da rodada 1 persiste: **OpenHarness** testa melhor *o harness* (121 arquivos por subsistema), **gemini-cli** testa melhor *o agente* (evals com juiz + baselines de regressão), **opencode** verifica melhor *o trabalho* (LSP em runtime → diagnósticos ao modelo no mesmo turno). Mas a lacuna reveladora da rodada 1 — "só um dos três testa comportamento sob ataque" — **fechou** na rodada 2: IronClaw trata isolamento cross-tenant como cidadão de teste de primeira classe (com *parity de trace* contra o OpenClaw), e ohmo tem 96 testes adversariais (sessão não vaza para outro remetente, `/config` não vaza segredos).

2. A verificação certa é externa e ancorada — porque a interna falha

O achado científico central (a auto-correção intrínseca degrada; a ancorada em tool funciona) é exatamente o que o **LSP em runtime** do opencode faz: o agente descobre que quebrou a tipagem no turno seguinte, não no CI. É a mesma tese da **reflexão do Aider** (disparada por lint/testes falhando, não por introspecção) e do **verify-on-stop do Hermes** — o agente é *forçado* a verificar antes de parar, com `verification_evidence.py` rastreando a evidência. Verificar deixou de ser uma esperança e virou um **estágio imposto do loop**.

3. Eval comportamental virou table-stakes — e por categoria

Na rodada 1, só o gemini-cli tratava comportamento como superfície de regressão. Na rodada 2 isso virou norma: o **Goose** publica a **Harbor** (sobre o framework Terminal-Bench, 89 tasks, com **leaderboard real**: stock 50,6% / code-mode 57,3%); o **Codex** tem ~660 snapshots insta; o **Hermes** roda `mini_swe_runner` (estilo SWE-bench); o **n8n** transformou eval em *produto* (nós Evaluation + LLM-judge). E surgiu o eval

por categoria: o **Personal Agent Benchmark Pack** do OpenClaw (10 cenários da categoria — `personal-redaction-no-secret-leak`, `personal-approval-denial-stop`, `personal-no-fake-progress`, `personal-memory-preference-recall`), o primeiro benchmark comportamental *da categoria agente pessoal*. Um harness sem evals não sabe o que perdeu no último ajuste de prompt.

4. O adversário é o próprio agente

A virada mais séria: a verificação virou **adversarial**. A literatura mostra agentes apagando asserts e patchando o pytest para "passar"; a defesa da indústria é convergente — **testes imutáveis** (commite os testes primeiro, o agente não os edita), **held-out/randomizado** (o agente não pode overfittar o que não vê), **política anti-mock** (o `AGENTS.md` de teste do opencode proíbe mocks que mentem; o `http-recorder` grava chamadas reais), e **snapshots com drift-check** (OpenClaw) para determinismo onde o juiz é caro. A verificação não é mais só medir acerto — é impedir a trapaça.

> **Adendo (2026-07-31, texto integral verificado): como avaliar o próprio harness — três regras de um paper de método.** O preprint *Rethinking the Evaluation of Harness Evolution for Agents* (AI2/UW/indep., 14-jul-2026) testa a moda da "evolução automática de harness" e encontra um resultado incômodo: sob **orçamento equiparado** (K=5 para todos os métodos), ela "does not consistently outperform simple test-time scaling methods" — no Terminal-Bench 2.1 (89 tarefas, 3 modelos), amostragem paralela pura levou a média de pass@1 de 68,2 a 72,3 (Tabela 1) enquanto a evolução chegou a **piorar** o GPT-5.4 (75,3→69,7); com testes unitários disponíveis, a amostragem paralela abre 86,0 contra 75,8 (Tabela 2); e em tarefas held-out o ganho médio da evolução é **+0,6** (Tabela 3) — "their gains largely stem from making multiple attempts" (§4.3), porque "most edits memorize fixes rather than distilling strategies" (§5.1), acumulando "context bloat that can offset the remaining gains". As três regras que ficam para quem avalia harness (incluindo este livro): (1) **orçamento equiparado** — todo ganho atribuído a design deve ser reportado contra um baseline de repetição de amostras com o mesmo compute; (2) **separação busca/avaliação** — held-out obrigatório, ou o ganho é overfitting ao conjunto; (3) **sensibilidade do instrumento** — os próprios autores suspeitam que "Terminal-Bench may simply not be very sensitive to harness design" (§5.2): benchmark bom para medir harness precisa de headroom E de desempenho que dependa do harness, senão o sinal é capacidade do modelo. Para o método deste livro (rubrica 0–3 por leitura de código), o paper refina sem contradizer: a rubrica mede a propriedade estrutural sem passar pelo canal contaminado por amostragem — mas herda o dever da **validade convergente** (notas altas deveriam prever desempenho held-out), o risco de overfitting se a régua for calibrada olhando os sistemas que se quer pontuar bem, e o alerta do §5.1: penalizar memorização e inchaço de contexto, não só ausência de recursos. Isso conversa com o cap. 16: se evoluir o harness automaticamente rende menos que re-amostrar, a auto-melhoria barata está no **conhecimento** (skills/memória), não na **estrutura**.

> **LEITURA EXECUTIVA**
>
> O que está mais moderno: verificação externa ancorada (LSP/testes no loop, verify-on-stop); eval comportamental como table-stakes e por categoria (Harbor, Personal Agent Benchmark Pack); o juiz LLM usado com controle de viés; a defesa contra reward hacking (testes imutáveis, held-out, anti-mock); e, para quem avalia o próprio harness, as três regras do adendo (orçamento equiparado, held-out, instrumento sensível a design). **O que roubar:** realimente sinal real ao modelo no mesmo turno (LSP/testes), não confie na auto-conferência; commite os testes antes e não deixe o agente editá-los; grade o estado final, não a última mensagem; e trate evals comportamentais como regressão de primeira classe.

Mão na massa — harness-zero, etapa 10

A etapa 10 (`harness-zero/etapas/10-evals/`) dá ao harness-zero uma suíte de evals própria: **respostas de LLM gravadas** (replay determinístico em CI, barato e estável) para testar o loop e as tools sem chamar a API, e um **juiz LLM** mínimo que pontua se o comportamento do agente atende a critérios qualitativos (usou a tool certa? respeitou o plan mode?). Fiel à disciplina do capítulo: o juiz emite um veredito discreto e a suíte roda no CI como qualquer teste. Exercício de completude: você adiciona um caso de **teste imutável** — uma tarefa cujo teste o agente é proibido de editar — e observa a diferença entre "passou" e "trapaceou".

Verificação

1. Seu agente diz "corrigi o bug e os testes passam". Por que isso, sozinho, não é verificação — e o que você faz em vez de confiar? (Auto-correção/auto-relato intrínseco não basta — 2310.01798; rode os testes reais e ocultos e grade por execução — SWE-bench.)

2. Depois de dar RLVR ao seu agente, o score sobe mas o produto piora. O que provavelmente aconteceu, e que duas defesas você aplica? (Reward hacking — o agente gameia o verificador, ex. apaga asserts; defesas: métrica held-out que ele não otimiza + testes imutáveis.)
3. Você usa um juiz LLM para pontuar respostas abertas. Cite um viés conhecido e como mitigá-lo. (Posição/verbosidade/self-preference; trocar a ordem e fazer a média, rubrica + gold set humano.)

Apêndice A — Como cada repositório trata verificação e evals

> Evidência por harness, com paths — complementação online, expandida a cada rodada.

gemini-cli (rodada 1) — comportamento sob regressão contínua

Quatro suítes: (1) `evals/` — ~45 testes comportamentais com **juiz LLM** (`llm-judge.ts`) cobrindo frugalidade, memória hierárquica, plan mode, delegação, segurança de shell, **prompt injection via MCP (Model Context Protocol)** e recuperação de sandbox; (2) `integration-tests/` — E2E determinísticos com **respostas gravadas** (`.responses`); (3) `memory-tests/` — regressão contra `baselines.json`, nightly; (4) `perf-tests/` — CPU/startup, nightly. Comportamento como superfície de regressão de primeira classe.

opencode (rodada 1) — verificação durante a tarefa

LSP em runtime (`packages/opencode/src/lsp/`): edições disparam diagnósticos realimentados ao modelo. **Política anti-mock** explícita (o `AGENTS.md` de `test/` proíbe mocks) + `http-recorder` (grava/replaya HTTP real com determinismo). Typecheck obrigatório (`bun typecheck`).

OpenHarness (rodada 1) — E2E com modelo real

121 arquivos em `tests/`, ~31 subpastas espelhando cada subsistema. Suítes E2E com **chamadas reais de modelo** (`scripts/test_harness_features.py`) e testes contra artefatos reais do ecossistema (`test_real_skills_plugins.py` roda skills do anthropics/skills e plugins do claude-code). Skill `harness-eval` empacota a validação E2E.

Goose (rodada 2) ⭐ — Harbor com leaderboard público

Harbor (`evals/harbor/`): benchmark sobre o framework Terminal-Bench (89 tasks) comparando harnesses/modelos/builds por pass-rate, custo, tokens e turns — com **leaderboard real no README** (stock ~50,6%, code-mode 57,3%) e LLM-judges de pós-processamento; `goose-self-test.yaml`; compactação com ~15 testes inline.

Codex CLI (rodada 2) — snapshots em escala

~440 arquivos de teste + **~660 snapshots insta**; suíte E2E com turnos reais e backend mockado; testes de política de sandbox por plataforma; parity da compactação remota; CI multi-camada (nextest por plataforma, Bazel, postmerge).

Hermes (rodada 2) ⭐ — verify-on-stop

32 subdiretórios de teste; **verify-on-stop nudge** (o agente é forçado a verificar antes de parar, com `verification_evidence.py` rastreando evidência); `batch_runner.py` (trajetórias em lote) e `mini_swe_runner.py` (avaliação estilo SWE-bench). Orientação a pesquisa.

OpenClaw (rodada 2) ⭐ — benchmark da categoria

~8.649 arquivos de teste; **prompt snapshots com drift-check** em CI; stack QA com canal sintético e catálogo YAML de cenários; **Personal Agent Benchmark Pack** — 10 cenários da categoria (`personal-redaction-no-secret-leak`, `personal-approval-denial-stop`, `personal-no-fake-progress`, `personal-memory-preference-recall`…), rodáveis em mock. O primeiro benchmark comportamental *da categoria agente pessoal*.

IronClaw (rodada 2) ⭐ — isolamento como cidadão de teste

~415 arquivos de teste; fuzzing; **testes de isolamento cross-tenant/agent/project/thread como primeira classe** (`reborn_*_scope_isolation_parity.rs`); **parity de trace gravado contra o OpenClaw**; testes de arquitetura mecanizados; regra que exige testes de denial/redaction/escape para qualquer mudança de sandbox.

ohmo (rodada 2) — adversarial de canal

96 testes adversariais (75 no gateway): sessão não restaura mensagens de outro remetente, `/config show` não vaza segredos, histórico de `/group` sanitizado antes de virar contexto. Lacuna: sem teste de permissão/sandbox — exatamente a dimensão fraca.

Aider (rodada 2) — reflexão ancorada + leaderboard de edit format

Reflexão (`reflected_message`, máx. 3) disparada quando o linter acha erros ou testes falham (sempre com confirmação humana) — auto-correção **reativa e ancorada**, não introspecção. Famoso por medir empiricamente o formato de edição por modelo (`percent_cases_well_formed`) num leaderboard próprio.

n8n (rodada 2) — eval como produto

Feature **Evaluations** (nós Evaluation Trigger + Evaluation, UI enterprise) para rodar datasets contra workflows; suíte de evals com LLM-judge no AI Workflow Builder; testes de integração por workflow. Verificação empacotada como recurso vendável.

OpenHands (rodada 2) — o eval que migrou

Eval de agente **ausente neste repo** (nota 0): o diretório `evaluation/` clássico (o harness SWE-bench pelo qual o OpenHands é histórico) migrou para o `software-agent-sdk`. Aqui há 115 arquivos de testes unitários do app-server, mas zero evals de agente — um lembrete de que a fronteira do que se avalia depende de onde o núcleo vive.

Frameworks (rodada frameworks)

Os frameworks tratam eval como API: harnesses de eval versionados (OpenAI Evals), Solver+Scorer com sandbox (Inspect), scorers mistos código+juiz (Braintrust/autoevals), rubrica-como-config (LangSmith). O que os harnesses de código montam à mão, o ecossistema de frameworks expõe como ferramenta dedicada.

12 — Extensibilidade

🕒 estado da arte 2026-07 · revisão 2026-07-26 · 📖 ~12 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Explicar** por que extensibilidade é "aberto para extensão, fechado para modificação" — extension points em vez de fork;
2. **Distinguir** os quatro eixos de extensão (hooks · comandos/skills · plugins · provedores) e o que cada um resolve;
3. **Comparar** as três estratégias de ecossistema — profundidade, empacotamento, interoperabilidade;
4. **Avaliar** o código de extensão como superfície de ataque (o *trust triangle*) e as defesas (scan, trust envelope, managed settings, least-privilege);
5. **Implementar** um subsistema de hooks pre/post-tool com o retorno do hook como canal de controle no harness-zero (etapa 11).

O problema

Nenhum harness cobre todos os fluxos de trabalho; a extensibilidade decide se o usuário **adapta** o harness ou o **abandona**. Os eixos consagrados:

1. **Hooks** — código do usuário interceptando o ciclo de vida (antes/depois de tool, compactação, sessão).
2. **Skills / comandos custom** — capacidades empacotadas como markdown/config, carregadas sob demanda.
3. **Plugins / extensions** — pacotes distribuíveis agregando tools, comandos, hooks e config.
4. **Provedores de modelo** — a extensão mais estratégica: o harness funciona com qualquer modelo, ou é vitrine de um?

A regra que une os quatro é antiga: **aberto para extensão, fechado para modificação** — o usuário estende sem editar (nem forkar) o core.

Fundamentos científicos

Registro editorial honesto (Princípio I): **não existe canon acadêmico de "extensibilidade de harness de agente"** — é uma lacuna real. As citações duráveis vêm da engenharia de software clássica de arquiteturas extensíveis e da segurança de ecossistemas de plugin, que transferem diretamente.

- **Extension points, não fork** — o princípio aberto-fechado (Meyer, 1988; Martin, 1996) e a arquitetura de plug-ins do Eclipse (Birsan, *ACM Queue* 2005) dão a fundação — e a advertência do "*plug-in hell*": pontos de extensão mal desenhados viram dívida. Decisão: exponha *seams* explícitos (eventos, diretórios conhecidos), não pontos ad-hoc.
- **Núcleo mínimo, extensões plugáveis** — o padrão Microkernel (Buschmann et al., *POSA* v.1, 1996) e sua encarnação agêntica, AIOS, arXiv 2403.16971 (um kernel que isola escalonamento/memória/tools das aplicações-agente), sustentam a postura "harness como microkernel": um core pequeno que serve de soquete.
- **Mecanismo × política** — Hydra (Levin et al., SOSP '75) é a origem de "separar mecanismo de política". Traduzido: o harness fornece o *mecanismo* (invocar tool, despachar hook, carregar provedor); a *extensão* fornece a política. É por isso que adicionar um provedor de modelo pode ser "escrever um arquivo".
- **Extensão de terceiros não é confiável** — a melhor citação on-topic é LLM (Large Language Model) Platform Security: ChatGPT Plugins, arXiv 2309.10254 (AIES '24): um *trust triangle* plataforma/plugin/usuário com exploits concretos (sequestro de sessão via plugin malicioso). E a base empírica de over-privilege vem da segurança de extensões de browser (Barth et al., NDSS '10: 88% das extensões pedem mais poder do que precisam). Decisão: least-privilege + isolamento + verificação — o mesmo argumento do *tool poisoning* do cap. 06.

(Bibliografia completa e ponteiros: `livro/bibliografia.md`.)

Fontes da indústria

- **Hooks: exit code como canal de controle** — os hooks do Claude Code expõem ~31 eventos de ciclo de vida (`PreToolUse`, `PostToolUse`, `Stop`, `SessionStart`, `UserPromptSubmit`, `PreCompact`, `SubagentStop`…) onde o harness executa comandos do usuário; o **exit code é o canal** (0 = segue / JSON no stdout com allow-deny-ask; 2 = bloqueia com stderr realimentado ao modelo). Decisão: times impõem política (bloquear `rm`, redigir `.env`, auto-lint) de forma **determinística e sem patchar o harness**. E o Codex implementa o mesmo padrão de forma independente (hooks + `allow_managed_hooks_only` para empresas) — hooks são padrão **cross-vendor**, não peculiaridade de um fornecedor.
- **Plugin = unidade de empacotamento; marketplace = catálogo** — o modelo de plugins do Claude Code: um plugin agrega skills, subagentes, hooks, MCP (Model Context Protocol) e LSP (Language Server Protocol) num pacote instalável (`/plugin install nome@marketplace`); um marketplace é um repo git com `.claude-plugin/marketplace.json`. Instala em escopo user/project/local/**managed**, com **pin a SHAs** e um modelo de confiança em dois níveis (marketplace oficial curado + comunidade com triagem de segurança). Decisão: extensão de terceiros vira distribuível **e governável** sem fork.
- **Comandos custom viraram file-drop (e AGENTS.md é o padrão aberto)** — no Claude Code, os comandos slash foram absorvidos pelas skills: largar um arquivo em `.claude/commands/` ou `.claude/skills/` cria o comando, sem registro nem build. E o AGENTS.md virou o formato de config **aberto e multi-tool** — lido por Codex, Cursor, Cline, Windsurf, Gemini CLI e Claude Code. Decisão: o ponto de extensão é "largue um arquivo num diretório conhecido", e o formato é portável entre harnesses.
- **Settings como superfície de enforcement** — a config do Claude Code é uma pilha de precedência (Managed > CLI > local > project > user); a maioria das chaves sobrescreve, mas **regras de permissão fazem merge**, e as **managed settings não podem ser sobrescritas** (uma equipe de segurança nega tools/marketplaces para toda a empresa). Decisão: config não é preferência, é enforcement (liga ao cap. 07).
- **Extensibilidade é também orçamento de contexto** — o advanced tool use (Anthropic) reenquadra: com bibliotecas ilimitadas de tools, a extensão precisa ser **carregada sob demanda**, não registrada de antemão; e plugins se ligam/desligam para controlar o custo de system prompt. Decisão: um ponto de extensão que sempre injeta contexto não escala — o carregamento tardio é parte do design (liga aos caps. 03 e 05).
- **Consulte também**: a coleção viva Awesome Harness Engineering — Debugging & Developer Experience reúne mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

1. Três estratégias de ecossistema

A moldura da rodada 1 persiste e ganhou reforço. **Profundidade**: os hooks alcançam pontos que os outros não expõem — o opencode transforma mensagens e system prompt antes do envio, intercepta `permission.ask` e registra provedores de auth. **Empacotamento**: a *extension* como unidade de distribuição completa (gemini-cli agrega MCP+comandos+hooks+políticas num pacote; Codex com manifest + marketplace + App Server JSON-RPC (Remote Procedure Call)). **Interoperabilidade**: adotar os formatos do líder em vez de inventar os próprios (OpenHarness com `SKILL.md`/`.claude-plugin`; IronClaw com `SKILL.md` compatível).

2. A aposta da interoperabilidade está vencendo — o "MCP da extensibilidade"

O que na rodada 1 era o eixo mais subestimado virou tendência dominante: os **formatos de extensão estão convergindo em padrões portáveis entre harnesses**. `SKILL.md`/AgentSkills (o OpenClaw usa o padrão `agentskills.io`; o IronClaw declara compatibilidade com OpenClaw/Claude), `.claude-plugin` (adotado pelo OpenHarness) e sobretudo o **AGENTS.md** (lido por seis harnesses diferentes) estão fazendo pela extensibilidade o que o MCP fez pela integração. Até o **vocabulário de hooks** convergiu — o conjunto de eventos do Codex é praticamente o do OpenHarness e o do Claude Code (`PreToolUse`/`PostToolUse`/… com decisões Approve/Block/Deny/Ask). A extensibilidade está deixando de ser silo por harness.

3. Marketplaces e scan de segurança — a lacuna da rodada 1 fechou

Na rodada 1, só o gemini-cli tratava código de extensão como superfície de ataque. Na rodada 2 isso virou norma, exatamente como o *trust triangle* de plugins previa: o **OpenClaw** tem o registry **ClawHub** com *trust envelope* + scan (VirusTotal/ClawScan); o Claude Code tem marketplace oficial curado + comunidade com triagem de segurança e **pin a SHA**; o **n8n** roda `scan-community-package`; o **Goose** verifica malware de extensões antes de carregar. Somado às **managed settings** que negam marketplaces enterprise-wide, a distribuição de extensões virou infraestrutura *com contenção* — o least-privilege que a literatura de over-privilege pede.

4. Provider-agnosticism virou config declarativa

A separação mecanismo × política aplicada ao modelo: adicionar um provedor deixou de ser código e virou arquivo. O **Goose** tem **37 provedores declarativos por JSON** (um provider OpenAI-compatible = um arquivo); o **opencode** tem ~26 loaders + centenas de modelos via models.dev; o **Hermes** tem `ProviderProfile` subclassável (Nous Portal com 300+ modelos). O harness agnóstico de modelo — que trata o provedor como política plugável — venceu a vitrine de um fornecedor só.

5. A próxima fronteira: o harness que se estende sozinho

O embrião da auto-extensão já aparece: o **IronClaw** tem **extração automática de skills** (`learning.rs`) com métricas de uso e confiança — o harness observa o próprio trabalho e escreve skills novas. É a ponte com o cap. 16 (aprendizado) e com a linhagem Voyager/ToolMaker: extensibilidade que não espera o usuário.

> **LEITURA EXECUTIVA**
>
> O que está mais moderno: a convergência de formatos (SKILL.md/.claude-plugin/AGENTS.md como padrões portáveis); marketplaces com scan de segurança e managed settings; hooks com exit-code como canal cross-vendor; provider-agnosticism declarativo; e o começo da auto-extensão. **O que roubar:** exponha seams explícitos (eventos nomeados, diretórios conhecidos) em vez de pontos ad-hoc; adote formatos portáveis em vez de inventar os seus; trate extensão de terceiros como não-confiável (scan + least-privilege + managed deny); e faça o carregamento ser tardio para não estourar o contexto.

Mão na massa — harness-zero, etapa 11

A etapa 11 (`harness-zero/etapas/11-hooks/`) dá ao harness-zero um subsistema de **hooks pre/post-tool**: antes de cada chamada de tool, hooks são funções registradas (`@hooks.pre_tool`/`@hooks.post_tool`) e o **retorno do hook é o canal de controle** (`"block:motivo"` bloqueia e realimenta o motivo ao modelo; um dict ajusta os argumentos) — o exercício de completude propõe a variante externa dos produtos: executar um comando do usuário e ler o exit code (0 segue; não-zero bloqueia com o stderr). É o mecanismo (o harness despacha o hook) separado da política (o usuário decide o que o hook faz) — a tese do capítulo em ~40 linhas. Exercício de completude: você adiciona um `PostToolUse` que roda um linter e devolve os erros ao modelo, e um gate de confiança mínimo (o hook só roda se o diretório for confiável).

Verificação

1. Por que "aberto para extensão, fechado para modificação" leva a *hooks* e *plugins* em vez de instruir o usuário a forkar o harness? (Extension points preservam o core e a atualizabilidade; o fork diverge e apodrece.)
2. Você vai permitir um marketplace de plugins de terceiros. Cite o risco central (com o nome da literatura) e duas defesas concretas. (*Trust triangle* / over-privilege; defesas: scan de segurança + pin a SHA + managed settings que negam + least-privilege.)
3. Seu harness precisa suportar um novo provedor de modelo sem release. Que princípio de design torna isso "escrever um arquivo"? (Separação mecanismo × política — o harness dá o mecanismo de invocação, o arquivo dá a política do provedor.)

Apêndice A — Como cada repositório trata a extensibilidade

> Evidência por harness, com paths — complementação online, expandida a cada rodada.

opencode (rodada 1) — hooks profundos e agnosticismo radical de provedor

Plugins são funções que retornam `Hooks` (`packages/plugin/`): **~15 pontos**, incluindo raros — transformar mensagens/system prompt antes do envio (`experimental.chat.messages.transform`), interceptar `permission.ask`, customizar compactação e **registrar provedores de auth** (`auth`). Tools custom auto-carregadas de `tool/`. E **~26 loaders de provedor** + centenas de modelos via models.dev, sobre o Vercel AI SDK (Software Development Kit) — o mais agnóstico de modelo em produção.

gemini-cli (rodada 1) — o pacote tudo-em-um

Extensions (`gemini-extension.json`): um pacote instalável agrega MCP servers, comandos custom, hooks, **políticas de permissão**, skills e temas. Comandos custom em TOML (`FileCommandLoader`). Hooks como subsistema (`packages/core/src/hooks/`) com **gate de confiança** (`trustedHooks.ts` — só rodam em pastas confiáveis). Provedores: ecossistema Google.

OpenHarness (rodada 1) — compatibilidade como estratégia

Skills em markdown carregadas também de `~/.claude/skills` e `~/.agents/skills` (layout `SKILL.md`); plugins no formato `.claude-plugin/plugin.json` (12 plugins reais testados); hooks cobrem **10 eventos** com **hot-reload**. Provedores como "workflows" nomeados (Anthropic/OpenAI-compatible, Copilot, Kimi, GLM, Ollama…).

Codex CLI (rodada 2) — hooks completos + marketplace + App Server

Hooks completos (`hooks/`: PreToolUse/PostToolUse/PreCompact/SessionStart-End/UserPromptSubmit/Stop/SubagentStart-Stop, decisões Approve/Block/Deny/Ask) e knob enterprise `allow_managed_hooks_only`; plugins com manifest e marketplace; skills; provedores configuráveis; profiles; SDKs Python/TS; **App Server JSON-RPC** como espinha dorsal programática.

OpenClaw (rodada 2) ⭐ — registry com scan de segurança

Skills no padrão **AgentSkills** (`agentskills.io`) com 6 níveis de precedência e registry público **ClawHub** com *trust envelope* + scan (VirusTotal/ClawScan); **159 plugins** (tools, canais, provedores, hooks, mídia) com Plugin SDK; dezenas de provedores LLM com failover e rotação de auth.

IronClaw (rodada 2) ⭐ — compatível e auto-extensível

Formato **`SKILL.md` compatível** com OpenClaw/Claude; skills v2 com snippets executáveis, métricas de uso/confiança e **extração automática de skills** (`learning.rs`); extensões via WASM/MCP/first-party **sem restart**; providers configuráveis (NEAR AI, Gemini OAuth…).

Goose (rodada 2) — provedores declarativos e distros brandeadas

Três eixos: extensões MCP (6 tipos de transporte/origem); recipes/skills; e provedores — nativos + **37 provedores declarativos por JSON** (adicionar um provider OpenAI-compatible = criar um arquivo). `CUSTOM_DISTROS.md` (distros brandeadas); `goose-sdk` para embutir; verificação de malware de extensões antes do carregamento.

Hermes (rodada 2) — ProviderProfile e plugins

`ProviderProfile` subclassável (**Nous Portal** com 300+ modelos sob assinatura, OpenRouter, endpoint próprio); sistema de plugins (20 diretórios, registry de toolsets, hooks de sessão); adaptadores Anthropic/Bedrock/Codex/ACP (Agent Client Protocol).

OpenHands (rodada 2) — marketplaces e injeção de dependência

Marketplaces de skills/plugins (instance/org/personal); LLM + agent profiles; camada de integrações Git plugável; agentes de terceiros via ACP; **backends de sandbox/event-store trocáveis por injeção de dependências**; litellm para provedores.

n8n (rodada 2) ⭐ — o catálogo como extensibilidade

O ponto mais forte: **os 400+ nós de integração viram pool de tools** sem escrever código (via `usableAsTool` + `$fromAI`); community nodes com scanner de segurança (`scan-community-package`); ~20 providers de modelo (`LmChat*`).

ohmo (rodada 2) — raízes extras

`~/.ohmo/skills` e `~/.ohmo/plugins` como raízes coexistindo com as do projeto; plugins carregam tools, slash commands e servidores MCP; skills viram comandos no canal; `channel_configs` arbitrário por canal.

Frameworks (rodada frameworks)

Os frameworks expõem extensibilidade como API: registro de tools/`@tool`, callbacks/hooks de ciclo de vida, adaptadores de provedor (litellm/model providers), e — cada vez mais — leitura do `AGENTS.md`. O formato portável (AGENTS.md, SKILL.md) é o que aproxima frameworks e harnesses de código num ecossistema comum.

13 — Interfaces

🕒 estado da arte 2026-07 · revisão 2026-07-26 · 📖 ~13 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Argumentar** por que "núcleo com front-ends" bate "front-end com um agente dentro" — e como desenhar a fronteira cedo maximiza as superfícies possíveis;
2. **Distinguir** as superfícies (TUI (Terminal User Interface), headless/SDK (Software Development Kit), IDE (Integrated Development Environment), chat, cloud) e o que cada uma exige do core;
3. **Avaliar** a UX de interação à luz da HCI (Human-Computer Interaction) (mixed-initiative, níveis de automação, over-reliance);
4. **Reconhecer** a superfície como fronteira de segurança (mesmo contrato de turn, não backdoor) e a virada para o paradigma *ambient*/inbox;
5. **Explicar** por que, com o loop atrás de portas, uma segunda superfície (headless) é um adapter fino, não uma reescrita (etapa 0 do harness-zero).

O problema

O mesmo agente precisa servir públicos diferentes: o desenvolvedor no terminal, o script de CI que precisa de JSON, o IDE que quer diffs inline, o gestor que acompanha por chat. A pergunta arquitetural é uma só: **o harness é um núcleo com múltiplos front-ends, ou um front-end com um agente dentro?** Os harnesses estudados responderam "núcleo com front-ends" — e a qualidade dessa separação determina quantas interfaces são viáveis.

Superfícies consagradas: **TUI interativa**, **headless/não-interativo** (`-p` com saída estruturada), **IDE** (diffs, contexto do editor), **CI/CD** (Actions), **protocolos de agente** (ACP (Agent Client Protocol), A2A (Agent-to-Agent)), **chat** (Slack, Telegram…) e, cada vez mais, **cloud/assíncrona**.

Fundamentos científicos

Registro editorial honesto (Princípio I): **não existe canon acadêmico de "interface de harness de agente"** — a lacuna é real. Mas a HCI de interação humano-IA a fundamenta com precisão, e um filete recente (2025-26) já trata de human-in-the-loop de agentes.

- **Quando agir × quando perguntar** — Principles of Mixed-Initiative UI (Horvitz, CHI '99): os 12 princípios sobre incerteza do objetivo, custo/benefício de agir e handoff gracioso *são* a decisão central de um harness — plan mode e aprovações (caps. 07/09) são "passar a iniciativa" aplicado.
- **O dial de autonomia é por estágio** — a escala de 10 níveis de automação (Sheridan & Verplank, 1978) e o modelo de tipos e níveis (Parasuraman, Sheridan, Wickens, 2000) mostram que a automação se aplica *independentemente* a cada estágio (aquisição · análise · decisão · ação). Decisão: o harness pode **auto-coletar contexto** (automação alta) e ainda **gatear a ação** (automação baixa) — o dial não precisa ser global.
- **A UX de "quando errar"** — Guidelines for Human-AI Interaction (Amershi et al., CHI '19): 18 diretrizes por fase; as de recuperação (correção/desfazer barato) explicam por que a reversibilidade (cap. 08) é também uma decisão de *interface*.
- **A supervisão é frágil — projete contra isso** — To Trust or to Think (Buçinca et al., CSCW '21) mostra que explicação sozinha **não** cura over-reliance; *forcing functions* cognitivas sim. A revisão de over-reliance (Passi & Vorvoreanu, MSR-TR-2022-12) sintetiza o risco. Decisão: a aprovação deve ser um **ato deliberado**, não um clique reflexo, e a superfície não pode esconder o que o agente fez. O trabalho recente de human-in-the-loop de agentes (Magentic-UI, arXiv 2507.22358, com *action guards* = gating de permissão; design de oversight, arXiv 2510.19512) operacionaliza isso.

(Bibliografia completa e ponteiros: `livro/bibliografia.md`.)

Fontes da indústria

- **Um núcleo, muitas superfícies (agora doutrina)** — a doc Platforms and integrations (Claude Code) diz explicitamente: "roda o mesmo motor subjacente em todo lugar, mas cada superfície é afinada para um jeito de trabalhar" (CLI, Desktop, VS Code, JetBrains, Web, Mobile + Chrome, GitHub Actions, GitLab, Slack), com **config, memória de projeto e MCP (Model Context Protocol) compartilhados** entre as superfícies locais. Decisão: construa o agente como um motor único e trate terminal/IDE/web/mobile como front-ends intercambiáveis.
- **Headless é um filtro Unix** — Run Claude Code programmatically (headless): `-p`/`--print`, `--output-format text|json|stream-json`, lê stdin e redireciona stdout "como qualquer ferramenta de linha de comando", com `--allowedTools`/`--permission-mode` para runs desatendidos nunca travarem num prompt. Decisão: a interface é stdin/stdout + exit codes — o agente cai em pipes, build scripts e CI sem UI.
- **O SDK é o loop empacotado; Managed Agents é o agente como serviço** — o Agent SDK dá "as mesmas tools, loop e gestão de contexto que movem o Claude Code", programável em Python/TS, e separa *quem roda o loop* (SDK no seu processo) de *quem o renderiza*; os Managed Agents levam ao extremo — "a Anthropic roda o agente e o sandbox, sua aplicação manda eventos e recebe o stream". Decisão: a superfície programática é o core como biblioteca — ou como endpoint REST.
- **O IDE é uma superfície fina sobre o mesmo motor** — VS Code e JetBrains adicionam diffs inline e contexto do editor reusando o engine do CLI (mesmo CLAUDE.md, mesmos modos de permissão); o padrão mais amplo — Copilot agent mode, Cursor 2.0 (background agents), Windsurf Cascade — divide a superfície do editor em **inline (síncrona)** e **background (assíncrona, cloud)** sobre a mesma abstração de tarefa.
- **A UX de interação: aprovação como máquina de estados, streaming como evento, humano como tool** — os modos de permissão fazem da aprovação uma máquina de estados (default/acceptEdits/plan/…), não um prompt ad-hoc; o streaming expõe o loop como stream tipado de eventos (`text_delta`, `tool_use`, `result`); e o AskUserQuestion modela o human-in-the-loop **como uma tool** que o agente chama — "perguntar ao humano" vira um passo do loop, não uma interrupção especial.
- **A virada ambient/inbox** — para agentes assíncronos, a superfície deixa de ser o prompt de chat e vira uma **caixa de entrada**: os ambient agents da LangChain (sempre-ligados, disparados por evento, que emergem ao humano só por notify/question/review) e o Claude Code na web (roda em cloud gerenciada e "continua depois que você desconecta") apontam o mesmo futuro: supervisionar *muitos* agentes de longa duração sem um terminal ao vivo — exatamente o "oversight sem oversight constante" que a HCI de níveis de automação prevê.
- **Chat como serviço, com identidade própria** — os channels deixam Telegram/Discord "ou seu próprio servidor" empurrar eventos para uma sessão; o Slack faz `@Claude` virar sessão cloud que transforma bug em PR, **com credenciais e trilha de auditoria próprias, desacopladas do acesso de qualquer humano**. Decisão: chat é só mais um gatilho, e o agente-serviço tem identidade própria (liga ao cap. 07).
- **Consulte também**: a coleção viva Awesome Harness Engineering — Human-in-the-Loop reúne mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

1. Núcleo com front-ends — a fronteira cedo decide tudo

A lição estrutural da rodada 1 virou consenso: **quanto mais cedo a fronteira núcleo/interface é desenhada, mais interfaces cabem depois**. O Codex é o exemplo cristalino — "um único motor Rust serve TUI, `codex exec` headless, extensão IDE, app desktop, cloud/web, servidor MCP e remote control". O opencode paga a mesma aposta com uma API HTTP tipada e clientes gerados. O anti-padrão é o inverso: um front-end com um agente enfiado dentro, que não escala para uma segunda superfície sem reescrita.

2. Headless com saída estruturada é obrigatório

Não há harness sério sem o modo filtro-Unix: `codex exec` (JSONL), gemini `--output-format stream-json` (NDJSON de eventos), `oh -p`/`ohmo --print`, Aider headless. A saída estruturada é o que torna o agente **programável** — peça de pipeline, alvo de CI, backend de outra UI. É a superfície que, uma vez ausente, fecha todas as outras automações.

3. Três visões — e a explosão do "colega no chat" com voz

As três apostas da rodada 1 persistem, agora mais nítidas: o agente como **produto** multi-plataforma (opencode Electron/VS Code; Codex desktop+cloud), como **serviço** de plataforma (gemini-cli SDK/A2A/Action; Managed Agents REST) e como **colega** no chat. A categoria de agentes pessoais *explodiu* a terceira: o **OpenClaw** serve **~23 canais** de chat + apps nativos (iOS/Android/macOS/Windows) + **voz** (Voice

Wake, Talk Mode contínuo) + Live Canvas; o **Hermes** tem um gateway multi-canal de processo único (10 plataformas + voz); o **ohmo** faz de Telegram/Slack/Discord/Feishu a superfície primária. A voz e a largura de canais viraram superfícies de primeira classe.

4. A superfície é fronteira de segurança, não backdoor

A lição mais madura da rodada 2, e a que a HCI de over-reliance reforça: uma superfície não pode ser um atalho em volta do core. O **IronClaw** materializa isso — CLI, WebUI, Slack, Telegram e webhooks entram todos pelo **mesmo contrato de turn** (`ProductAdapter`), e a WebUI é *proibida* de bypassar as fronteiras de auth. O agente-no-Slack roda com credenciais e auditoria próprias, desacopladas do humano. E a UX precisa não *esconder* o que o agente fez — o antídoto contra a falsa sensação de supervisão que Buçinca e Passi & Vorvoreanu documentam.

5. A próxima fronteira: ambient, cloud, assíncrono

O paradigma emergente muda a própria natureza da interface. O **Codex cloud-tasks** (TUI de tarefas remotas), o Claude Code na web que continua após desconectar, e o inbox dos ambient agents apontam para o mesmo lugar: o humano deixa de *dirigir* um agente ao vivo e passa a *supervisionar muitos* por notificação e revisão. É o dial de autonomia da HCI levado ao produto — automação alta na execução, o humano no gate de decisão, assíncrono. A interface do agente está saindo do terminal (o watch mode do Aider transforma comentários `ai!` em qualquer editor; o Live Canvas do OpenClaw) e virando ambiente.

> **LEITURA EXECUTIVA**
>
> O que está mais moderno: um motor, muitas superfícies (doutrina); headless estruturado obrigatório; a explosão de canais + voz; a superfície como fronteira de segurança (mesmo contrato de turn); e a virada ambient/inbox/cloud. **O que roubar:** desenhe a fronteira núcleo/interface cedo (API tipada ou biblioteca), não tarde; entregue headless com `stream-json` desde o dia um; faça toda superfície passar pelo mesmo contrato de turn (nunca um backdoor de auth); modele o human-in-the-loop como tool e a aprovação como ato deliberado; e prepare-se para o inbox — o próximo terminal é assíncrono.

Mão na massa — harness-zero: o chat como janela de observação

A interface do harness-zero nasceu na **etapa 0**: um chat mínimo sobre FastAPI, a *janela de observação* que acompanha cada etapa do livro. A lição desta dimensão é o que o projeto inteiro demonstra: porque o loop vive atrás de portas (`LLMPort`, `ToolPort`, `StorePort`), acrescentar uma **segunda superfície** — um modo headless `--print` que emite os mesmos eventos em `stream-json` — é um **adapter fino**, não uma reescrita. Exercício de completude: você adiciona o modo headless e prova que o mesmo agente responde no chat e no pipe, e que a aprovação (o gate de permissão do cap. 07) aparece nas duas superfícies pelo mesmo contrato — a superfície não é backdoor.

Verificação

1. Por que "núcleo com front-ends" permite mais interfaces do que "front-end com um agente dentro", e o que decide isso na prática? (A fronteira desenhada cedo — API tipada/biblioteca — deixa cada superfície ser um adapter fino; o inverso exige reescrita por superfície.)
2. Seu agente vai rodar assíncrono em cloud, supervisionado por vários humanos. Que paradigma de interface e que princípio de HCI guiam o design? (Ambient/inbox — notify/question/review; níveis de automação: alta na execução, humano no gate de decisão; over-reliance → não esconder o que o agente fez.)
3. Você expõe o agente no Slack e numa WebUI. Que regra impede que a nova superfície vire um furo de segurança? (Mesmo contrato de turn/ProductAdapter — toda superfície passa pelas mesmas fronteiras de auth; a UI não bypassa; identidade/auditoria próprias.)

Apêndice A — Como cada repositório trata as interfaces

> Evidência por harness, com paths — complementação online, expandida a cada rodada.

opencode (rodada 1) — a maior superfície de produto

Arquitetura cliente-servidor (cap. 02): servidor HTTP com API tipada e clientes gerados habilitam **sete superfícies** — TUI (SolidJS/opentui), **app desktop Electron** (única na rodada 1), extensão VS Code, **GitHub Action** (`packages/github/`), **Slack** (`packages/slack/`), web (`packages/web/`) e **ACP** (integração Zed). Sessões compartilháveis por link conectam as superfícies.

gemini-cli (rodada 1) — terminal rico + plataforma

TUI React/Ink com ~40 slash commands por loaders plugáveis. **Headless de primeira classe**: `gemini -p` com `--output-format stream-json` (NDJSON em tempo real). **VS Code companion** (servidor IDE expondo arquivos/diffs). GitHub Action oficial. **ACP** para editores e **A2A server**. SDK próprio (`packages/sdk`).

OpenHarness/ohmo (rodada 1) — o agente que mora no chat

CLI Typer (`oh`) headless (`-p`, `text|json|stream-json`) + `--dry-run`; duas TUIs (React/Ink + Textual); dashboard web do autopilot. E o **ohmo**: agente pessoal em **Telegram/Slack/Discord/Feishu** (`channels/` + `gateway/`) com workspace próprio.

OpenClaw (rodada 2) ⭐ — a maior largura de superfície

~23 canais (WhatsApp, Telegram, Slack, Discord, Signal, iMessage, Teams, Matrix, Feishu, LINE, WeChat, QQ…), Control UI web, WebChat, CLI, TUI, **voz** (Voice Wake + Talk Mode contínuo), **apps nativos** (iOS/Android/macOS/Windows) e **Live Canvas** (A2UI). A superfície do agente como produto de consumo.

Codex CLI (rodada 2) ⭐ — um motor, todas as superfícies

Um único motor Rust serve: TUI (ratatui), `codex exec` headless (humano + JSONL), extensão IDE via App Server, **app desktop**, **cloud/web** (`cloud-tasks` com TUI de tarefas remotas), Codex como servidor MCP e remote control. O exemplo canônico de núcleo com front-ends.

IronClaw (rodada 2) ⭐ — mesmo contrato de turn

CLI/REPL, WebUI (SSE+WS com OIDC, rate limit, origin check), Slack, Telegram, webhooks — todos entrando pelos **mesmos contratos de turn** (`ProductAdapter`); a WebUI é **proibida de bypassar** as fronteiras de auth. A superfície como fronteira de segurança, não backdoor.

Hermes (rodada 2) — gateway multi-canal de processo único

TUI completa; **gateway multi-canal**: Telegram, Discord, Slack, WhatsApp, Signal, Email, iMessage, QQ, WeChat, Yuanbao — com continuidade cross-plataforma; **voz** (transcrição + TTS multi-provider); ACP para editores; servidor API OpenAI-compatível.

Goose (rodada 2) — desktop ACP sobre core embarcado

CLI completo + TUI; **desktop Electron falando ACP** com o core (binário embarcado, sem servidor separado); headless via recipes + scheduler; gateway Telegram e bot Discord; modo servidor MCP/ACP puro.

Aider (rodada 2) — input fora do terminal

CLI/REPL rica (prompt_toolkit, streaming markdown), browser UI (Streamlit), **watch mode** (`aider/watch.py`: comentários `ai!`/`ai?` no código de qualquer IDE viram comandos), **voz-para-código**, imagens/URLs no chat. A interface escapando para o editor de terceiros.

OpenHands (rodada 2) — control-plane SaaS

Web UI React (~40 rotas: conversas, settings, admin, billing, orgs); CLI `agent-canvas`; headless/REST via Agent Server; **resolvers GitHub/GitLab/Jira/Slack** (webhooks); enterprise/SaaS completo (Keycloak, Stripe, multi-tenant); deploy Docker/k8s.

n8n (rodada 2) — chat embarcável + canvas

Chat Trigger (app de chat hospedado + widget `@n8n/chat` embarcável + streaming), Manual Chat Trigger, webhooks arbitrários, editor visual (canvas) como interface de construção, MCP Server Trigger. O "harness invertido" cuja interface primária é o grafo.

Frameworks (rodada frameworks)

Os frameworks entregam o loop como biblioteca (a superfície programática pura) + streaming de eventos + human-in-the-loop como composição (OpenAI Agents SDK, LangGraph, CrewAI); a UI fica a cargo do integrador. É o extremo "só núcleo, superfície é sua" do espectro — o oposto do OpenClaw.

14 — Convergências e Tendências

🕒 estado da arte 2026-07 · revisão 2026-07-28 · 📖 ~7 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Enumerar** as oito convergências arquiteturais da primeira rodada e **explicar** por que convergência independente sinaliza disciplina consolidada;
2. **Distinguir** as dimensões consolidadas das dimensões em divergência real, e **justificar** por que a contenção é a divergência mais consequente;
3. **Aplicar** a cláusula de expiração a um componente de harness qualquer — identificando por que ele existe e sob que condição expira;
4. **Avaliar** um harness novo contra o checklist de convergências, cobrando justificativa para cada ausência;
5. **Antecipar** as tendências a acompanhar nas próximas rodadas e o que cada uma implicaria para o desenho de harnesses.

O problema

Os capítulos anteriores analisaram o harness dimensão por dimensão — contexto, compactação, tools, permissões, loop. Falta a pergunta que dá sentido ao conjunto: **o que é acidente de implementação e o que é anatomia da disciplina?** Sem essa síntese, cada capítulo é um catálogo de escolhas; com ela, o leitor ganha um critério de projeto — saber o que copiar sem hesitar, onde ainda cabe apostar diferente, e o que vai desaparecer quando os modelos melhorarem.

O instrumento de medida é a convergência independente. Quando equipes que não se coordenam, em stacks e culturas diferentes, chegam à mesma arquitetura, isso é evidência forte de que o problema — e não a moda — determinou a solução. E o instrumento de projeção é a cláusula de expiração do capítulo 01: todo componente de harness é uma prótese para uma limitação atual do modelo, e portanto todo componente deve declarar quando espera se tornar desnecessário.

O estado da arte

O achado central da primeira rodada: oito convergências

Três harnesses, três stacks (Effect-TS, TypeScript, Python), três origens (startup independente, big tech, academia/porta didática) — e uma convergência arquitetural notável. Sem coordenação, os três chegaram a:

1. **Arquivo de contexto hierárquico na raiz do projeto** — `AGENTS.md` / `GEMINI.md` / `CLAUDE.md`: o mesmo artefato com três nomes (cap. 03).
2. **Compactação em escada** — truncar tools → prune → sumarizar via LLM, com disparo automático por limiar (cap. 04).
3. **Schema de tools derivado de tipos** — Effect Schema, classes declarativas, Pydantic: ninguém escreve JSON Schema à mão (cap. 05).
4. **MCP como integração padrão** — três clientes completos sobre os SDKs oficiais (cap. 06).
5. **Plan mode como modo de permissão** — read-only imposto pelo sistema de permissões, não pedido ao modelo (cap. 09).
6. **Hooks de ciclo de vida** — before/after tool, compactação, sessão (cap. 12).
7. **Headless com saída estruturada** — `-p` + JSON/NDJSON para scripting e CI (cap. 13).
8. **Parada por ausência de tool-call + limite de turnos** — a mecânica universal do loop (cap. 02).

Quando implementações independentes convergem assim, a anatomia está consolidada: **isto é a disciplina**, não mais um conjunto de escolhas idiossincráticas. Um harness novo que não implemente os oito itens acima precisa justificar cada ausência.

Onde ainda há divergência real

As dimensões sem consenso são o mapa das apostas em aberto:

- **Contenção** (cap. 07): política + sandbox de SO obrigatórios (gemini-cli), política + paths sensíveis fixos (OpenHarness), ou só política (opencode)? A divergência mais consequente — é a que define o risco operacional.

- **Multi-agente** (cap. 10): ferramenta pontual, serviço com registry, ou time persistente com mailbox? Três filosofias incompatíveis; o vencedor depende de quão bons os modelos ficarão em coordenação.
- **Quem decide continuar** (cap. 02): heurística estrutural ou uma inferência extra por turno (next-speaker check)?
- **Neutralidade de modelo** (cap. 12): ~26 provedores (opencode) contra vitrine de um ecossistema (gemini-cli). Aposta comercial, não técnica — mas define quem sobrevive à comoditização dos modelos.
- **Evals comportamentais** (cap. 11): na rodada 1, só um dos três tratava comportamento do agente como superfície de regressão — a rodada 2 confirmou a previsão e a lacuna fechou (ver cap. 11). Previsão fácil: em dois anos, isso será tão obrigatório quanto CI.

A cláusula de expiração, aplicada

Retomando a tese do capítulo 01 — todo componente de harness é uma prótese para uma limitação atual do modelo. O exercício que todo harness deveria fazer, aplicado ao que estudamos:

Componente	Existe porque...	Expira quando...
Compactação	janelas são finitas e caras	contexto longo ficar barato e confiável
Plan mode	modelos agem precipitadamente	modelos planejarem espontaneamente sob risco
Next-speaker check	o modelo não sinaliza bem o fim do turno	protocolos de turno nativos do modelo
Policy engine / aprovações	modelos não são confiáveis com ações destrutivas	confiabilidade calibrada e verificável
Prompt por família de modelo	modelos respondem diferente a instruções	convergência de instruction-following
Subagente para exploração	dumps de arquivos poluem o contexto	contexto abundante + atenção robusta
Repo-map / índices de código	o modelo não "carrega" o repo inteiro	contexto de milhões de tokens utilizável

O que **não** expira: sandbox (contenção é sobre o mundo, não sobre a capacidade do modelo), interfaces, verificação do trabalho (testes/LSP — verdade externa ao modelo), e os protocolos de interoperabilidade (MCP, A2A, formatos de skill). A engenharia de harness de longo prazo mora aí: **na fronteira entre o agente e o mundo, não na muleta para a limitação do modelo**.

Tendências a acompanhar nas próximas rodadas

1. **Padronização do arquivo de contexto** — a pressão por `AGENTS.md` neutro cross-vendor.
2. **Skills/plugins portáveis** — o OpenHarness já carrega skills do formato Claude Code; um "MCP da extensibilidade" está se formando.
3. **Agente-como-serviço** — A2A server, agent cards, SDKs: harnesses expondo-se uns aos outros.
4. **Segurança como dimensão de primeira classe** — parsing de shell, trusted folders, evals de injection: hoje exceção, amanhã baseline (hipótese confirmada na rodada 2 com o Codex CLI).
5. **Reversibilidade** — checkpoint git com `/rewind`: quando desfazer é barato, a política pode ser mais frouxa; espere mais harnesses copiando.
6. **O harness mínimo** — na contramão da sofisticação, projetos como mini-swe-agent (~100 linhas) testam quanto do *scaffolding* (andaime) o modelo moderno já dispensa. É a cláusula de expiração virando experimento.

LEITURA EXECUTIVA

- Oito dimensões já convergiram entre implementações independentes — são o checklist mínimo de um harness sério; ausências exigem justificativa.
- As divergências reais (contenção, multi-agente, next-speaker, neutralidade de modelo, evals comportamentais) são o mapa das apostas em aberto — contenção é a de maior consequência operacional.
- A cláusula de expiração separa próteses temporárias (compactação, plan mode, repo-map...) do que é permanente: sandbox, interfaces, verificação externa e protocolos de interoperabilidade.
- O valor de longo prazo da engenharia de harness está na fronteira agente–mundo; o resto muda de dono ou desaparece conforme os modelos melhoram.
- Este capítulo é o placar vivo do livro: cada rodada do benchmark confirma convergências, resolve divergências ou aposenta componentes expirados.

> **Consulte também**: a coleção viva Awesome Harness Engineering — Foundations reúne mais recursos consultáveis desta dimensão, curados por problema.

Verificação

1. Por que a convergência **independente** (três stacks, três origens) é evidência mais forte de consolidação do que a adoção de um padrão por vários projetos que se copiam? (Releia "O problema" e o achado central.)
2. Um harness novo não implementa plan mode nem arquivo de contexto na raiz. Segundo este capítulo, qual é a postura correta ao avaliá-lo — e o que você exigiria do autor?
3. Aplique a cláusula de expiração a um componente que **não** está na tabela (por exemplo, o next-speaker check já está; escolha hooks de ciclo de vida ou headless): ele existe por limitação do modelo ou por necessidade da fronteira agente–mundo? Ele expira?
4. Entre as cinco divergências listadas, qual define o risco operacional e qual é uma aposta comercial em vez de técnica? Justifique com o texto.

15 — O Harness Embutido

🕒 estado da arte 2026-07 · revisão 2026-07-28 · 📖 ~8 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Explicar** a inversão que define a categoria — o workflow contém o harness, e não o contrário — e por que ela levanta a pergunta "quais dimensões do scaffolding são essenciais, e quais são substituíveis pelo ambiente?";
2. **Identificar** quais dimensões do scaffolding o ambiente de workflow dispensa (compactação, planejamento, entrega de contexto, permissões granulares) e **justificar** por que cada uma se torna dispensável;
3. **Analisar** a implementação de um nó de agente real (o AI Agent do n8n, Apêndice A como gabarito) e localizar onde o loop, as tools e as permissões vivem;
4. **Avaliar** quando usar um harness embutido versus um dedicado, em função da duração e da autonomia da tarefa — e reconhecer o teto da substituição;
5. **Aplicar** as ideias exportáveis da categoria a um harness dedicado: derivação de tools a partir de superfícies existentes (padrão `$fromAI`) e human-in-the-loop durável.

O problema

Nos capítulos anteriores, o harness contém o trabalho: o loop dirige, as tools agem, o workflow emerge das decisões do modelo. Ferramentas como o **n8n** invertem a relação — **o workflow contém o harness**. Um "nó de agente" é uma etapa dentro de um grafo desenhado por um humano, cercado de gatilhos (webhook, cron, chat), integrações e tratamento de erro que o motor de workflow já fornecia antes de existir IA.

Essa inversão levanta a pergunta que dá sentido à categoria: **quais dimensões do scaffolding são essenciais, e quais são substituíveis pelo ambiente?**

O estado da arte

O que o ambiente dispensa

A avaliação do representante da categoria (n8n, ver Apêndice A) confirma a tese com precisão incômoda: as dimensões fracas do harness embutido são exatamente as que o ambiente dispensa.

Dimensão dispensada	Por que o ambiente dispensa
Compactação	Execuções acionadas por evento são curtas — o contexto não acumula
Planejamento	O plano *é* o grafo do workflow, desenhado pelo humano no canvas
Entrega de contexto	O contexto vem mapeado das etapas anteriores via expressões
Permissões granulares	A topologia já é a allowlist

O último ponto merece ênfase: no harness embutido, **a permissão é topologia**. Não há aprovação por chamada dentro do loop — o LLM (Large Language Model) só pode invocar o que o autor plugou no canvas. É allowlist por construção, decidida visualmente por um humano, complementada por human-in-the-loop real: nós que pausam a execução de forma durável aguardando aprovação num canal (Slack/Outlook), em vez do prompt síncrono de aprovação dos CLIs.

E as dimensões fortes são onde o motor tem vantagem estrutural: **ferramentas** (as integrações pré-existentes viram pool de tools), **memória** (backends de banco plugáveis), **interfaces** (chat hospedado, webhooks, widget embarcável), **MCP (Model Context Protocol)** (client *e* server) e **subagentes** (agente-como-tool e sub-workflows). Nenhum harness dedicado tem um pool de tools do tamanho de um ecossistema de integrações convertido — porque nenhum tem um ecossistema pré-existente para converter.

O loop emprestado — e a trajetória de reinternalização

O harness embutido tipicamente não escreve o próprio loop: ele o toma emprestado de um framework (no caso observado, LangChain JS). Mas a trajetória medida no benchmark aponta numa direção clara: o motor de workflow começa terceirizando o loop e **reinternaliza a metade que importa para um motor de workflow — o agendamento da execução**. O framework continua decidindo *qual* tool chamar; a *execução* da chamada volta a ser responsabilidade do engine, que agenda os nós e reentra no agente. (Detalhe de código no Apêndice A, achado 1.) A implicação: os motores tendem a absorver cada vez mais o harness, não o contrário.

O teto da substituição

Mas a substituição tem teto: **sem compactação nem planejamento, o nó de agente serve automações curtas, não trabalho longo autônomo**. Um agente embutido que precisasse refatorar um repositório por horas colapsaria a janela de contexto sem defesa. As duas camadas não competem — se complementam por duração e autonomia da tarefa: o harness dedicado para trabalho longo e aberto; o embutido para decisões pontuais dentro de processos estruturados.

Implicações

1. **Para quem constrói harness dedicado**: o padrão `$fromAI` (Apêndice A, achado 2) mostra como derivar tools de superfícies existentes sem escrever wrappers; o HITL durável (pausar a execução por dias aguardando aprovação num canal) é superior ao prompt síncrono de aprovação dos CLIs.
2. **Para quem constrói sobre motores de workflow**: as lacunas da categoria (compactação, plan mode) são o roadmap óbvio — e a trajetória de reinternalização do loop sugere que os motores vão absorver cada vez mais o harness, não o contrário.
3. **Para a taxonomia do livro**: "quanto harness é preciso" é função do *ambiente de execução*, não constante universal. A régua do benchmark mede scaffolding presente; esta categoria lembra que scaffolding ausente-por-design não é lacuna — desde que a classe de tarefa seja respeitada.

LEITURA EXECUTIVA

O harness embutido não é um harness dedicado incompleto: é uma categoria em que o ambiente de execução substitui, por construção, metade das dimensões do scaffolding — plano vira grafo, permissão vira topologia, contexto vira expressão mapeada. A substituição vale enquanto a classe de tarefa for respeitada: decisões pontuais dentro de processos estruturados, não trabalho longo autônomo. **O que roubar** hoje: derivação automática de tools a partir de integrações existentes (padrão `$fromAI`) e human-in-the-loop durável em vez de aprovação síncrona.

> **Consulte também**: a coleção viva Awesome Harness Engineering — Production Infrastructure & Operations reúne mais recursos consultáveis desta dimensão, curados por problema.

Verificação

1. Enuncie a inversão que define a categoria e explique por que ela transforma "dimensões fracas" do benchmark em "dimensões dispensadas pelo ambiente". (Se precisar, releia "O que o ambiente dispensa".)
2. Por que a permissão-como-topologia dispensa aprovação por chamada dentro do loop — e qual mecanismo complementa essa allowlist quando uma decisão humana é realmente necessária no meio da execução?
3. Um time quer usar um nó de agente de motor de workflow para refatorar um repositório por horas. Explique, em termos de compactação e planejamento, por que isso colapsa — e qual seria a divisão correta entre harness embutido e dedicado nessa tarefa.
4. Cite as duas ideias da categoria que valem exportação para um harness dedicado e o que cada uma substitui ou melhora. (Dica: derivação de tools e HITL.)

Apêndice A — n8n (nó AI Agent)

> Evidência por repositório, com paths — material de complementação (versão online), expandido a cada rodada do benchmark. A

> avaliação completa do n8n (29/36) está em `../../benchmark/avaliacoes/n8n.md`.

Anatomia do nó de agente (evidência: `packages/@n8n/nodes-langchain`)

O n8n implementa o agente como um "cluster node": um nó-raiz **AI Agent** com portas tipadas onde se plugam sub-nós — modelo (`AiLanguageModel`), memória (`AiMemory`), ferramentas (`AiTool`), parser de saída. Três achados de código estruturam o capítulo:

1. O loop é emprestado — e está sendo devolvido. A geração V2 delega tudo ao LangChain JS (`AgentExecutor.fromAgentAndTools`, `maxIterations` 10). Mas a V3 mudou o desenho: o LangChain ainda *decide* qual tool chamar (`createToolCallingAgent`), porém a *execução* virou responsabilidade do motor do n8n — as tool calls viram `EngineRequest` devolvidos ao engine, que agenda os nós e reentra no agente com `EngineResponse`. O n8n começou terceirizando o loop e está **reinternalizando** a metade que importa para um motor de workflow: o agendamento da execução.

2. A ponte `$fromAI` — a ideia mais exportável da categoria. `create-node-as-tool.ts` transforma **qualquer um dos 400+ nós de integração** marcado `usableAsTool` numa tool do agente: o traversal dos parâmetros coleta expressões `$fromAI('chave', 'descrição', tipo)` — os slots que o LLM deve preencher — e gera o schema Zod automaticamente. Nenhum harness dedicado tem um pool de tools desse tamanho, porque nenhum tem um ecossistema de integrações pré-existente para converter.

3. A permissão é topologia. Não há aprovação por chamada dentro do loop: o LLM só pode invocar o que o autor plugou na porta `AiTool` do canvas. É allowlist por construção, decidida visualmente por um humano — complementada por human-in-the-loop real (nós `sendAndWait` pausam a execução de forma durável aguardando aprovação no Slack/Outlook, proibidos dentro de subagentes) e um nó Guardrails.

O placar (29/36) e o mapa força/fraqueza

As dimensões fracas da avaliação são as que o ambiente dispensa: **compactação (1)** — execuções acionadas por evento são curtas, o contexto não acumula; **planejamento (1)** — o plano é o grafo desenhado no canvas; **entrega de contexto (2)** — o contexto vem mapeado das etapas anteriores via expressões; **permissões granulares (2)** — a topologia já é a allowlist.

E as fortes são onde o motor tem vantagem estrutural: **ferramentas (3)** — as integrações; **memória (3)** — backends de banco plugáveis; **interfaces (3)** — chat hospedado, webhooks, widget embarcável; **MCP (3)** — client **e** server: o `McpTrigger` expõe as tools do n8n a clientes MCP externos; **subagentes (3)** — agente-como-tool e sub-workflows.

Primos a avaliar em rodadas futuras: Zapier Agents, Make, Dify, Flowise.

16 — Aprendizado e Auto-melhoria

🕒 estado da arte 2026-07 · revisão 2026-07-28 · 📖 ~8 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Explicar** por que o aprendizado auto-evolutivo quebra o pressuposto do *scaffolding* estático — e como ele inverte a cláusula de expiração do livro;
2. **Descrever** as etapas do ciclo fechado de captura de skills (gatilho, curadoria, isolamento, formato portável, reencontro indexado, manutenção contra a entropia);
3. **Comparar** os dois designs concorrentes de aplicação do aprendizado — autônoma × promoção humana — e **localizar** um harness real na escada de maturidade da dimensão;
4. **Avaliar** os riscos da dimensão (superstição, entropia, contaminação, prompt injection como aprendizado permanente) e as engenharias que os previnem.

O problema

As doze dimensões dos capítulos 02–13 descrevem *scaffolding* (andaime) *estático*: alguém — o autor do harness, o usuário, um plugin — escreve as instruções, tools e políticas, e o agente as consome. Este capítulo documenta a dimensão emergente que quebra esse pressuposto: o agente que **escreve o próprio scaffolding** — capturando procedimentos aprendidos como skills reutilizáveis.

A dimensão foi promovida a suplementar do template do benchmark (dimensão 13) por força de uma evidência: o **Hermes Agent** (Nous Research) implementa o ciclo completo, e a leitura do código confirma cada etapa (Apêndice A).

O estado da arte

O ciclo fechado: as seis etapas

O mecanismo de referência, verificado no código do Hermes (evidência detalhada no Apêndice A), fecha o ciclo em seis etapas:

1. **Gatilho autônomo** — a revisão de aprendizado dispara sozinha, em background, sem o usuário pedir (com gatilho manual como complemento).
2. **Curadoria por um fork isolado** — um clone do agente, com um prompt curatorial que define o que capturar e — o mais importante — **anti-padrões do que NÃO aprender**. Sem essa lista, o sistema degeneraria em superstição acumulada.
3. **Isolamento do meta-trabalho** — o fork curador tem tools restritas e persistência desligada, para não contaminar a sessão real.
4. **Escrita em formato portável** — a skill vira um `SKILL.md` sob standards rígidos, com a restrição de contexto moldando o formato do conhecimento.
5. **Reencontro barato** — índice compacto sempre no system prompt; conteúdo integral só entra no contexto sob demanda. Aprendizado indexado, não despejado.
6. **Manutenção contra a entropia** — um curador periódico consolida, arquiva por inatividade e protege o que está fixado. Memória que só cresce vira ruído; o curador é o coletor de lixo do conhecimento.

A escada de maturidade na coorte avaliada

Harness	Nota 13	O que tem
Hermes	**3**	O ciclo fechado completo (Apêndice A), com aplicação autônoma
gemini-cli	**3** (retro)	Auto Memory: agente extrator com gates anti-ruído ("Default to NO SKILL", 5 perguntas de bloqueio) produzindo SKILL.md + patches de memória — mas com **promoção humana via inbox** (`/memory inbox`); dedupe, sandbox de escrita, evals dedicados
IronClaw	2	Extração automática de skills (`learning.rs`) com métricas de uso/confiança e versionamento
OpenClaw	1	Dreaming (consolidação autônoma de memória); Skill Workshop com fila de propostas
OpenHarness	1 (retro)	Auto-extração de fatos por turno, com staleness por uso (60 dias) — fatos, não procedimentos
Codex CLI	1	Memórias automáticas com pruning (fatos, não procedimentos)
Goose	1	chatrecall (recall semântico de conversas passadas)
opencode, demais	0 (retro)	Skills são consumo/distribuição; nada é escrito pela experiência

A escada é nítida: **memória de fatos** (nível 1) → **extração de procedimentos** (nível 2) → **ciclo curado com anti-padrões e manutenção** (nível 3). O que separa o nível 3 não é capturar mais — é a engenharia de *não* capturar errado e de podar o que envelheceu.

Os dois designs concorrentes do nível 3

O nível 3 já tem **dois designs concorrentes**, com a divergência exatamente onde importa: *quem aplica o que foi aprendido*. O Hermes aplica autonomamente (com o curador limpando depois); o gemini-cli exige promoção humana (inbox — nada entra no contexto sem `/memory inbox`). É o trade-off clássico autonomia × controle do capítulo 07, reaparecendo na dimensão mais nova: o Hermes aposta que anti-padrões bastam para prevenir aprendizado ruim; o gemini-cli aposta que não. As próximas rodadas dirão qual escala melhor.

Por que isso muda a tese do livro

A cláusula de expiração (cap. 01, 14) diz: todo componente de harness é uma prótese para uma limitação atual do modelo, e expira quando o modelo melhora. O aprendizado auto-evolutivo **inverte a cláusula**: em vez de esperar o modelo dispensar o scaffolding, o par modelo+harness *escreve scaffolding novo para si mesmo*. Cada skill aprendida é um pedaço de harness gerado em runtime, específico ao usuário e ao ambiente — algo que nenhum autor de harness poderia ter escrito de fábrica.

Isso cria uma terceira via na taxonomia:

1. **Scaffolding de fábrica** — escrito pelo autor do harness; expira com a evolução dos modelos.
2. **Scaffolding de fronteira** — sandbox, permissões, interfaces; não expira (é sobre o mundo).
3. **Scaffolding auto-gerado** — skills escritas pelo agente; *cresce* com o uso, e sua qualidade depende da engenharia de curadoria, não da capacidade bruta do modelo.

Os riscos: o espelho das promessas

Os riscos são o espelho das promessas: sem anti-padrões, superstição; sem curadoria, entropia; sem isolamento do meta-trabalho, contaminação; e — apontado pela avaliação do IronClaw (prompt-write safety; cf. cap. 07) — sem fronteira de escrita protegida, **prompt injection vira aprendizado permanente**: um atacante que convence o agente a "aprender" uma skill maliciosa persiste na memória procedural. A dimensão 13 madura exigirá a dimensão 6 madura.

> **LEITURA EXECUTIVA**
>
> A dimensão é a mais nova do template e a menos convergida: dois harnesses no nível 3 com designs opostos sobre quem aplica o aprendizado, e o resto da coorte entre memória de fatos e nada. O que já é consenso de engenharia entre os que chegaram lá: a peça central não é o mecanismo de captura, e sim os **anti-padrões do que não aprender** e a **manutenção** (consolidar, arquivar, nunca deletar). **O que roubar** hoje: lista de anti-padrões no prompt curatorial; isolamento do meta-trabalho em fork sem persistência; índice compacto com conteúdo sob demanda; curador periódico como coletor de lixo; fronteira de escrita protegida contra prompt injection.
>
> *Reavaliação retroativa da coorte de código pendente; a dimensão sai de "suplementar" quando ≥3 harnesses atingirem nível 2+.*

> **Consulte também**: a coleção viva Awesome Harness Engineering — Skills & MCP reúne mais recursos consultáveis desta dimensão, curados por problema.

Verificação

1. Por que a lista de **anti-padrões** ("o que NÃO aprender") é descrita como a peça central da engenharia curatorial, e não o mecanismo de captura em si? O que acontece com um sistema que captura sem ela?
2. Localize na escada de maturidade um harness que extrai fatos automaticamente com staleness por uso, mas não captura procedimentos. Que nota ele recebe, e o que faltaria para subir um nível?
3. Hermes e gemini-cli estão ambos no nível 3, mas divergem em *quem aplica* o que foi aprendido. Reconstrua o trade-off autonomia × controle nesse contexto: qual é a aposta de cada design?
4. Explique a frase "a dimensão 13 madura exigirá a dimensão 6 madura": por que prompt injection é qualitativamente mais grave num harness que aprende do que num harness estático?

Apêndice A — Hermes Agent

> Evidência por repositório, com paths — material de complementação (versão online), expandido a cada rodada do benchmark. Avaliação completa: `../../benchmark/avaliacoes/hermes-agent.md`.

O ciclo fechado do Hermes (evidência: `agent/background_review.py` e afins)

O mecanismo, verificado no código do fork avaliado:

1. Gatilho autônomo. A cada ~10 iterações de tool-calling (`skill_nudge_interval`, em `agent/turn_finalizer.py`), o harness dispara uma revisão em background — sem o usuário pedir. Há também o gatilho manual `/learn`.

2. Curadoria por um fork isolado. Um clone do agente roda em thread separada com o snapshot da conversa e um prompt curatorial (`_SKILL_REVIEW_PROMPT`) que é a peça central da engenharia. Ele instrui o curador a ser ativo ("um passe que não faz nada é aprendizado perdido"), define ordem de preferência (atualizar skill existente > criar nova; skills novas só class-level, nunca "fix-bug-1234") e — o mais importante — lista **anti-padrões do que NÃO aprender**: falhas dependentes de ambiente, claims negativos sobre tools ("o browser não funciona"), erros transitórios, narrativas one-off. Sem essa lista, o sistema degeneraria em superstição acumulada.

3. Isolamento do meta-trabalho. O fork tem whitelist de tools restrita (`memory` + `skills`), memória e persistência desligadas — para a curadoria não contaminar a sessão real — e herda o prefixo de prompt cacheado do pai (redução de ~26% no custo da revisão).

4. Escrita em formato portável. A skill vira um `SKILL.md` compatível com **agentskills.io** em `~/.hermes/skills/<categoria>/<nome>/` (com `references/`, `templates/`, `scripts/`), sob standards rígidos — descrição ≤60 caracteres *porque o índice no system prompt trunca em 60*: a restrição de contexto moldando o formato do conhecimento.

5. Reencontro barato. O índice compacto (nome + descrição) está sempre no system prompt; o conteúdo integral só entra no contexto quando o agente chama `skill_view` — aprendizado indexado, não despejado.

6. Manutenção contra a entropia. Um **curador** periódico (`agent/curator.py`) roda quando o agente está ocioso: consolida skills em umbrellas, arquiva por inatividade (90 dias — arquivar, nunca deletar), protege skills fixadas. Memória que só cresce vira ruído; o curador é o coletor de lixo do conhecimento.

17 — A Camada de Protocolos

🕒 estado da arte 2026-07 · revisão 2026-07-31 · 📖 ~9 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Explicar** por que a camada de protocolos é o que transforma um mercado de silos em um ecossistema — e por que cada protocolo padroniza uma *fronteira* diferente do harness;
2. **Distinguir** as fronteiras cobertas por MCP (Model Context Protocol), A2A (Agent-to-Agent) e ACP (Agent Client Protocol), agentskills.io e AGENTS.md — incluindo as duas confusões clássicas (os dois "ACP"; MCP × A2A como vertical × horizontal);
3. **Analisar** a matriz de adoção medida no código e localizar um harness real nela;
4. **Avaliar** a saúde de um protocolo por adoção medida e governança (fundação neutra × vendor único), em vez de por marketing;
5. **Decidir** quais protocolos um harness novo precisa falar para não ficar fora das arquiteturas de composição dos outros.

O problema

Os capítulos 02–16 tratam do que acontece *dentro* de um harness. Este capítulo trata do que acontece *entre* eles — e entre harnesses e o resto do mundo. Sem protocolos compartilhados, cada harness é um silo: suas ferramentas, suas instruções de projeto, seus subagentes e suas skills só funcionam dentro dele. A camada de protocolos é o que transforma esse mercado de silos em um ecossistema: cada protocolo padroniza uma fronteira diferente do harness — agente ↔ ferramenta, agente ↔ agente, agente ↔ editor, agente ↔ usuário, além dos formatos transversais de conhecimento procedural (SKILL.md) e de instruções de projeto (AGENTS.md).

A consequência prática: em um mercado que *compõe* harnesses, não falar os protocolos não é perder uma feature — é ficar de fora das arquiteturas dos outros.

O estado da arte

O mapa: um protocolo por fronteira

O mapa, organizado pela fronteira que cada um resolve:

Protocolo	Fronteira	Origem / governança	Estado (2026)
MCP (Model Context Protocol)	agente ↔ **ferramentas/dados**	Anthropic → adoção universal (OpenAI, Google, Microsoft)	maduro; ~97M downloads
A2A (Agent-to-Agent)	agente ↔ **agente** (delegação entre organizações)	Google → **Linux Foundation** (v1.0 em 2026)	consolidando; absorveu o ACP (Agent Communication Protocol) da IBM
ACP (Agent Client Protocol)	agente ↔ **editor/cliente**	Zed	adoção rápida entre harnesses de código
agentskills.io (Agent Skills / SKILL.md)	**conhecimento procedural** portável	Anthropic (spec aberta, dez/2025)	~40 produtos compatíveis em 6 meses
AGENTS.md	**instruções de projeto** portáveis	comunidade → **Agentic AI Foundation** (Linux Foundation)	60.000+ repositórios; 20+ ferramentas leem nativamente
AG-UI	agente ↔ **interface de usuário**	comunidade (CopilotKit)	emergente
ACP-IBM (Agent Communication Protocol)	agente ↔ agente	IBM	**encerrado** — fundido ao A2A (ago/2025)

Duas confusões a desfazer: (1) "ACP" designa dois protocolos distintos — o da IBM (comunicação agente-agente, descontinuado em favor do A2A) e o da Zed (agente-editor, vivo e em expansão); neste livro, ACP = Zed. (2) MCP e A2A não competem: MCP é a conexão *vertical* (agente→ferramenta), A2A é a *horizontal* (agente→agente peer) — um sistema real usa os dois.

A matriz de adoção — medida no código, não no marketing

O diferencial deste capítulo: cruzamos os protocolos com as **11 avaliações de harnesses do benchmark** (mais os 4 frameworks da rodada frameworks-1) (evidência por arquivo, ver `benchmark/avaliacoes/`). Nenhum comparativo externo tem esta coluna de verdade:

Harness	MCP client	MCP server	ACP	A2A	SKILL.md / agentskills	AGENTS.md (ou equiv.)
opencode	✅	—	✅ (Zed)	—	parcial	✅ AGENTS.md
gemini-cli	✅	—	✅	✅ **client+server**	✅	GEMINI.md
OpenHarness	✅	—	—	—	✅ (formato Claude)	CLAUDE.md
Codex CLI	✅	✅	—	—	✅	✅ AGENTS.md
Goose	✅	✅ (`goose mcp`)	✅ (desktop inteiro)	—	✅	✅ AGENTS.md + .goosehints
Aider	❌	❌	—	—	—	✅ (leitura)
OpenHands	✅	✅ (FastMCP)	✅ (perfis)	—	✅ (repos org)	microagents
OpenClaw	✅	✅	✅ (orquestra terceiros)	—	✅ (52 bundled)	✅ AGENTS.md + SOUL.md
Hermes	✅	✅	✅	—	✅ (núcleo do learning)	✅ AGENTS.md + SOUL.md
IronClaw	✅	—	—	—	✅ (compat OpenClaw)	identity files
n8n	✅	✅ (Trigger)	—	—	—	—
frameworks:						
LangGraph	❌	❌ (só no servidor pago)	❌	❌	❌	—
OpenAI Agents SDK (Software Development Kit)	✅	—	❌	—	parcial	só sandbox agents
CrewAI	✅ (obrigatório)	—	✅ **client+server**	—	✅	✅ **auto-gerado**
software-agent-sdk	✅ (OAuth)	—	❌	✅ (usa harnesses como motor)	✅ (spec)	✅

Leituras da matriz:

1. **MCP venceu de fato**: 10 de 11 (a exceção, Aider, é escolha filosófica). E entre as rodadas 1 e 2, o padrão migrou de "cliente" para "cliente+servidor" — o harness como serviço consumível.
2. **agentskills.io é a padronização mais rápida que já medimos**: spec de dezembro/2025, 8 dos nossos 11 compatíveis em julho/2026. A previsão do cap. 12 ("um MCP da extensibilidade está se formando") se cumpriu — e com um detalhe estrutural: skills são markdown portável, então a mesma skill roda no Claude Code, no Hermes e no IronClaw. O aprendizado auto-evolutivo (cap. 16) escreve *nesse* formato — o conhecimento que um agente aprende é, em tese, transferível a outro.
3. **ACP é o protocolo silencioso mais importante da coorte**: 6 de 11 o falam, e três harnesses (OpenClaw, OpenHands, Goose) o usam para **orquestrar outros harnesses** como subagentes — Claude Code, Codex, Gemini CLI e opencode viram peças intercambiáveis. O que era "agente↔editor" virou, na prática, o barramento de composição entre harnesses.
4. **A2A saiu do "aposta de um só"** *(atualizado na rodada frameworks-1)*: o gemini-cli foi o único harness a implementá-lo, mas o **CrewAI** entrou com client E server nativos (AgentCard completo, JWS, gRPC/REST) — o segundo implementador medido, e o primeiro framework. A governança na Linux Foundation e a absorção do ACP-IBM seguem apontando o A2A como o candidato à fronteira inter-organizacional; nos harnesses de produto, porém, essa fronteira ainda quase não existe.

5. **AGENTS.md consolidou como padrão neutro**: a fragmentação AGENTS/CLAUDE/GEMINI.md do cap. 03 está se resolvendo — Codex, Goose, opencode, OpenClaw e Hermes já convergiram para AGENTS.md (agora sob a Agentic AI Foundation), com os arquivos proprietários virando alias.

O empilhamento: como os protocolos compõem

Um sistema agêntico completo em 2026 usa a pilha inteira, uma camada por fronteira:

```
[usuário]
   │  AG-UI / canais de chat / TUI          (interface)
[harness A]
   │  ACP                                    (composição: A dirige B como subagente)
[harness B]
   │  A2A                                    (delegação a agente de outra organização)
[agente remoto]
   │  MCP                                    (cada agente alcança suas ferramentas)
[ferramentas/dados]

transversais: AGENTS.md (instruções por projeto) · SKILL.md (procedimentos portáveis)
```

Implicações para a engenharia de harness

1. **Protocolo é dimensão de sobrevivência, não de feature**: o Aider, referência técnica em três dimensões, está fora do ecossistema de composição inteiro por não falar MCP/ACP. Em um mercado que compõe harnesses, não falar os protocolos é ficar de fora das arquiteturas dos outros.
2. **A cláusula de expiração não se aplica aqui** (cap. 14): protocolos são fronteira com o mundo — o scaffolding que *resta* quando os modelos melhoram. Investir em protocolo é o investimento de harness com maior meia-vida.
3. **Para o benchmark**: a matriz acima vira seção permanente do comparativo, atualizada a cada rodada. Protocolos não recebem nota 0–3 como harnesses — são avaliados por **adoção medida** (a matriz) e **saúde de governança** (fundação neutra > vendor único).

Adendo (2026-07-31): a spec MCP **2026-07-28** (anúncio) reforça a tese deste capítulo por outro ângulo: núcleo stateless, framework de extensões e a **primeira política formal de depreciação** (12 meses) são o comportamento típico de protocolo saindo da adolescência e entrando na fase de infraestrutura — versionamento disciplinado importa mais que features. A adoção da nova versão pela coorte entra na matriz na próxima rodada. E a conferência do mesmo dia na outra fronteira (spec 065): a especificação do A2A confirma o **v1.0 estável sob a Linux Foundation**, organizado em três camadas (modelo de dados em Protobuf/JSON Schema, operações abstratas, bindings JSON-RPC/gRPC/REST), com o **v1.0.1 já trazendo um mecanismo formal de extensões** — os dois vencedores de fronteira chegaram, no mesmo trimestre, ao mesmo estágio: extensões formais em vez de features no núcleo.

> **LEITURA EXECUTIVA**
>
> A camada de protocolos já tem um vencedor por fronteira: MCP na vertical (agente→ferramenta, adoção quase total), ACP como barramento de composição entre harnesses, agentskills.io como formato portável de conhecimento procedural e AGENTS.md como padrão neutro de instruções de projeto — enquanto o A2A segue como a aposta em consolidação para a fronteira inter-organizacional, sustentada mais pela governança (Linux Foundation, absorção do ACP-IBM) do que pela adoção medida nos harnesses de produto. A decisão de engenharia é assimétrica: protocolos são o componente de maior meia-vida do harness, imune à cláusula de expiração, e a matriz de adoção — não o marketing — é o instrumento para reavaliá-los a cada rodada do benchmark.

Fontes da indústria

- ecosystem map 2026
- Zylos: convergência MCP/A2A/ACP
- Zuplo: onde foi parar o ACP
- Agent Skills: formato e adoção
- AGENTS.md guide 2026
- Zed ACP

Matriz de adoção: evidência própria do benchmark (`benchmark/avaliacoes/`).

- **Consulte também**: a coleção viva Awesome Harness Engineering — Skills & MCP reúne mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

Verificação

1. Um colega afirma que "o A2A vai substituir o MCP". Por que a afirmação confunde as fronteiras, e como o empilhamento mostra que um sistema real usa os dois? (Releia "O mapa" e o diagrama.)
2. "ACP" aparece duas vezes na tabela de protocolos, com estados opostos ("adoção rápida" e "encerrado"). Explique a diferença entre os dois protocolos — e qual deles este livro chama de ACP.
3. Você está desenhando um harness novo. Com base nas leituras da matriz e nas implicações, quais protocolos são obrigatórios hoje, qual ainda é aposta, e o que a exceção do Aider ensina sobre o custo de não falar nenhum?
4. Por que a cláusula de expiração (cap. 14) não se aplica à camada de protocolos, quando se aplica a quase todo o resto do harness?

Benchmark

Comparativo Consolidado — Rodadas 1, 2, ext-1, ext-2 e ext-4

> 16 harnesses avaliados por leitura sistemática de código, 12 dimensões (0–3) + 2 suplementares. Rodada 1: 2026-07-24 (opencode, gemini-cli, OpenHarness). Rodada 2: 2026-07-24 (Codex CLI, Goose, Aider, OpenHands, OpenClaw, Hermes, IronClaw, n8n). Rodada **ext-1**: 2026-07-31 (**Grok Build**, **Pi**). Rodada **ext-2**: 2026-08-02 (**Kimi Code**, **QM** — este inaugurando a categoria *agentes organizacionais*). Rodada **ext-3**: 2026-08-02 (**Traycer** — avaliado e **não incluído**). Rodada **ext-4**: 2026-08-06 (**Prime Agent**). Ver metodologia.

Categoria: harnesses de código

#	Dimensão	opencode	gemini-cli	OpenHarness	Codex CLI	Goose	Aider	OpenHands*	Grok Build	Pi	Kimi Code	Prime Agent
1	Loop	3	3	2	3	3	2	2	3	3	3	3
2	Contexto	3	3	2	3	3	**3**	3	3	3	3	3
3	Compactação	3	3	3	3	3	2	2	3	**3**⭐	3	3
4	Ferramentas	2	3	3	3	3	3	2	3	3	3	3
5	MCP	3	3	2	3	3	**0**	3	3	**0**	2	2
6	Permissões/sandbox	2	3	2	**3**⭐	2	2	3	**3**⭐	1	2	**1**
7	Memória/estado	3	3	3	3	3	3	3	3	2	2	3
8	Planejamento	2	3	2	2	2	2	1	3	1	3	2
9	Subagentes	2	3	3	3	3	2	2	**3**⭐	1	3	**3**⭐
10	Verificação/evals	2	3	2	3	3	3	0*	2	3	2	2
11	Extensibilidade	3	3	3	3	3	3	3	**3**⭐	**3**⭐	3	3
12	Interfaces	3	3	2	3	3	3	3	3	3	**3**⭐	3
	Total	**31**	**36**	**29**	**35**	**34**	**28**	**27***	**35**	**26**	**32**	**31**
13	**Aprendizado** (supl.)	—	3	—	—	—	—	—	—	2	1	**3**⭐

* OpenHands: o repo avaliado é o control-plane (Agent Canvas); o núcleo (loop, condenser, evals SWE-bench) migrou para `software-agent-sdk` — o total subestima o projeto completo. O SDK entra na fila.

Leitura da rodada ext-1 (2026-07-31):

1. **Os extremos do espectro chegaram juntos.** O Grok Build (35) empata com o Codex CLI cobrindo tudo com profundidade industrial — inclusive lendo os artefatos dos concorrentes (AGENTS/CLAUDE/Cursor/`.mcp.json`) e portando as tools do codex e do opencode. O Pi (26) pontua 3 em **tudo que aceita** e 0–1 em tudo que recusa por manifesto — o perfil serrilhado não é imaturidade, é tese ("adapt pi to your workflows, not the other way around"), e cada exclusão existe como extensão de exemplo testada.
2. **A dimensão 6 continua separando produto de projeto** — e o Grok Build sobe a régua: autorização de shell por **AST** (tree-sitter-bash), fecho do bypass `mv secret x && cat x`, sandbox kernel-enforced fail-closed. O Pi é o contraexemplo deliberado (1): terceiriza o boundary ao SO e argumenta que sandbox in-process é teatro.
3. **Evals comportamentais seguem sendo o gap mais comum** — o Grok Build tem 26k testes de mecanismo e zero de competência (nota 10 = 2); o Pi, na contramão, é o único da rodada com bancada de A/B de configurações de harness (`evalHarnessTable`) e artefatos de eval no formato nativo de sessão.

Leitura da rodada ext-2 (2026-08-02):

1. **O segundo vendor verticalizado confirma o padrão — e a divergência.** O Kimi Code (32) repete o movimento do Grok Build (modelo próprio → harness próprio, aberto), mas com aposta oposta: onde o xAI foi a plataforma máxima em Rust com sandbox kernel-enforced, a Moonshot foi de **autonomia estruturada** — goal mode com máquina de estados e budgets (turns/tokens/tempo), swarm de até 128 subagentes, cron exposto ao modelo — sobre enforcement fraco (sem sandbox de SO; bash autorizado por glob de string no engine em produção, com o parser AST pronto mas só no v2 experimental). O detalhe que nenhum outro tem: **co-design harness ↔ API** — a API do Kimi ganhou a capability `dynamically_loaded_tools` para servir a *progressive tool disclosure* do harness, com degradação documentada para outros provedores. O vendor mudou o modelo para servir o harness.
2. **Polinização cruzada dentro do corpus virou rotina**: a TUI do Kimi Code é um fork vendorizado da `pi-tui` (agradecimento no README); o QM traz Pi, OpenCode, Codex e Claude Code como *motores* plugáveis. O corpus deixou de ser uma lista de concorrentes e virou uma cadeia de suprimentos.
3. **Evals comportamentais seguem sendo o divisor** também na ext-2: o Kimi Code tem 1.137 arquivos de teste de mecanismo e zero evals de competência; o QM, na contramão, roda **E2E multiplayer contra Slack real com juiz LLM** — a implementação mais completa da dimensão 10 fora do gemini-cli.

Leitura da rodada ext-4 (2026-08-06):

1. **O anúncio e o código discordam — e o código venceu.** O lançamento afirma que *"fixed tool-calling schemas and context compaction force the model to work around its own scaffolding"*. Na leitura, a compactação **não foi eliminada nem enfraquecida**: as 1.398 linhas de `core/compaction/` herdadas do Pi continuam lá, com corte seguro, split turns e recuperação reativa de overflow — e ainda **melhoradas** (instruções customizadas, recálculo de `tokensBefore`). O que mudou é a **subordinação**: `compact.run()`/`compact.status()` viraram chamáveis pelo agente, com handler que **agenda em vez de executar** (executar abortaria a célula que pediu), rodando mesmo com auto-compaction desligada, sob 12 testes. A Leitura executiva do cap. 04 **não cai — ganha ressalva**: compactação deixou de ser evento involuntário do harness e virou um de vários mecanismos contra o crescimento de contexto, além de virar gatilho de destilação (`autoRefine.compact: true` por padrão).
2. **O Pi virou substrato do corpus.** O Prime Agent é o **Pi por baixo** — mesmos quatro pacotes (`pi-agent-core`, `pi-ai`, `pi-coding-agent`, `pi-tui`), LICENSE com dupla titularidade (Mario Zechner + Prime Intellect). É o **quinto** consumidor do Pi no mapa da cadeia de suprimentos, e o mais radical: colapsou o *tool set* inteiro num **único tool `ipython`** e construiu por cima o REPL como superfície de programa. A ironia fecha o argumento do cap. 12: o sistema de **menor nota do corpus** (Pi, 26) — que recusa metade das dimensões por manifesto — foi a base escolhida por um laboratório de fronteira. A extensibilidade do Pi recebeu sua validação empírica mais forte.
3. **A dimensão 13 tem um novo teto — e a 10 um novo piso.** O *Continual Harness* dá ao agente **CRUD do próprio estado** (prompt, memória, skill, spec de subagente) com escopo local/global, review gate a cada 25 turnos e rollback por snapshot: supera o Hermes e vira a referência da dimensão 13. Mas a **assimetria tese ↔ medição é gritante**: o repositório **não contém nenhum eval** — o `packages/evals` do Pi *desapareceu no fork* —, o `expectedOutcome` que o próprio `/refine` emite não é validado por nada, e a alegação de 95,5% no ARC-AGI-3 não tem artefato reprodutível no código. Um harness que se auto-modifica sem bancada de medição é a combinação mais arriscada que o benchmark já registrou.
4. **Regressões herdadas para baixo.** Além dos evals, o **Project Trust do Pi foi removido** (grep por `trust` em `src/` → nada) e o tool único é execução de Python arbitrário, com filhos herdando cwd e permissões — daí o **1** na dimensão 6, abaixo do já baixo piso do Pi.

Categoria: agentes organizacionais *(nova na ext-2)*

#	Dimensão	QM
1	Loop	3
2	Contexto	**3**⭐
3	Compactação	3
4	Ferramentas	3
5	MCP	1
6	Permissões/sandbox	3
7	Memória/estado	3
8	Planejamento	1
9	Subagentes	2
10	Verificação/evals	**3**⭐
11	Extensibilidade	3
12	Interfaces	3
	Total (1–12)	**31**
13	**Aprendizado** (supl.)	2
14	**Proatividade** (supl.)	**3**⭐

O QM (Y Combinator) inaugura a categoria: o primeiro harness do corpus em que a unidade de design é a **organização**, não a sessão de um usuário — escopos (pessoa/time/sala/org), contexto filtrado por *entitlement* de toda a audiência presente (`context-filter.ts`), consentimento de destinatário para entregas autônomas e auditoria como primitivas do core. O loop do agente é uma **dependência trocável** (Pi, OpenCode, Codex ou Claude Code por configuração), com a sessão portável entre motores via "fita" re-semeável. É a tese da commoditização do loop escrita em `package.json` — e a razão de a categoria ser nova: nas dimensões clássicas ele pontua como um harness maduro (31/36), mas o que o define não cabe nelas.

Categoria: agentes pessoais self-hosted

#	Dimensão	OpenClaw	Hermes	IronClaw	ohmo[1]
1–5	Loop/Contexto/Compact./Tools/MCP	3,3,3,3,3	3,3,3,3,3	3,3,3,3,3	3,3,3,3,3
6	Permissões/sandbox	3	3	**3**⭐⭐	2
7	Memória/estado	3	3	3	3
8	Planejamento	3	2	2	2
9	Subagentes	3	3	2[2]	3
10	Verificação/evals	3	3	3	3
11	Extensibilidade	3	3	3	3
12	Interfaces	3	3	3	3
	Total (1–12)	**36**	**35**	**34**	**34**
13	**Aprendizado** (supl.)	1	**3**⭐⭐	2	2
14	**Proatividade** (supl.)	3	2	3	3

[1] avaliação dedicada (2026-07-24) do app pessoal do OpenHarness — gap concentrado na dim. 6 (config de permissão/sandbox do gateway é código morto; sem dial entre nega-tudo e full_auto). [2] design nota-3, mas `spawn_subagent` está desabilitado em produção.

Categoria: harnesses embutidos

n8n (nó AI Agent)	**Total 1–12: 29/36**	**Fortes: tools 3 (`$fromAI` → Zod sobre 400+ integrações), MCP 3 (client+server), memória 3, subagentes 3, interfaces 3 · Fracas por design do ambiente: compactação 1, planejamento 1, contexto 2, permissões 2 (estrutural/topológica)**

Categoria: frameworks de harness (rodada frameworks-1, template FRAMEWORK_EVAL)

Eixo	LangGraph	OpenAI Agents SDK	CrewAI	software-agent-sdk
A1 Loop/orquestração	3	3	3	3
A2 Estado/durabilidade	**3⭐⭐**	3	3	3
A3 Tools/schemas	2	3	3	3
A4 Multi-agente	2	3	3	3
A5 Human-in-the-loop	3	**3⭐**	3	3
A6 Streaming/eventos	3	3	3	3
Total A (0–18)	**16**	**18**	**18**	**18**
D1 Observabilidade	2	2	2	2
D2 Testes/evals	3	3	3	3
D3 Ergonomia	2	3	3	3
D4 Ecossistema	3	3	3	3
Total D (0–12)	**10**	**11**	**11**	**11**

Leitura da rodada frameworks-1:

1. **As primitivas viraram commodity** (A quase todo 3) — a diferenciação real está nos eixos B (fronteiras) e C (protocolos), que são descritivos: LangGraph impõe BSP e deixa contexto/permissões totalmente abertos; Agents SDK impõe o vocabulário Responses; CrewAI impõe a ontologia papel/tarefa; o SDK da OpenHands impõe o modelo de eventos inteiro.
2. **Nenhum framework tem observabilidade aberta first-class** (D1=2 em todos): cada um gravita para sua plataforma (LangSmith, OpenAI, AMP, Laminar) — o espaço do "OTel de agentes" segue vago.
3. **Protocolos separam os campos**: CrewAI (MCP obrigatório + **A2A client/server** + skills + AGENTS.md auto-gerado) e software-agent-sdk (MCP OAuth + **ACP** + agentskills) são os poliglotas; o Agents SDK fala só MCP; **LangGraph fala zero** — protocolos são feature do servidor pago.
4. **A previsão dos "dois movimentos" confirmou-se no código**: o software-agent-sdk é o harness-virando-framework mais avançado (tudo virou ABC plugável, e seu `ACPAgent` orquestra Claude Code/Gemini/Codex como motores); o LangGraph faz o movimento oposto — **esvaziando-se** da camada de agente (create_react_agent deprecado rumo ao pacote langchain) para ser só runtime durável.
5. **Compactação continua sendo a linha divisória harness/framework**: só o software-agent-sdk a entrega pronta (condenser com tombstones — o melhor medido no benchmark inteiro); LangGraph/Agents SDK/CrewAI deixam a janela de contexto por conta do usuário (Agents SDK tem apenas uma session de compactação; CrewAI nada).

Leitura executiva da rodada 2

As hipóteses registradas na rodada 1 foram confrontadas — 3 confirmadas, 1 surpresa:

1. ✅ **Codex CLI = novo teto em contenção** (35/36): Seatbelt + bubblewrap/seccomp + Landlock + execpolicy Starlark + network-proxy — três camadas independentes. O gemini-cli deixa de ser o único "3 de referência" na dimensão 6.
2. ✅ **Goose = MCP-nativo confirmado** (34/36): até as tools internas são servidores MCP reais servidos in-process. O empate técnico Codex/Goose/gemini-cli no topo da categoria código indica que a fronteira de produto está convergindo.
3. ✅ **Aider = o caminho alternativo em contexto** (28/36): repo-map (tree-sitter + PageRank) é referência em entrega de contexto sem loop de agente — e o primeiro **0** do benchmark (MCP) mostra o custo da filosofia.

4. ⚠️ **OpenHands = surpresa metodológica** (27/36*): o repo virou control-plane; o núcleo está num SDK externo. Lição: a unidade de avaliação precisa acompanhar a decomposição dos projetos.

A categoria agentes pessoais estreou com nível inesperadamente alto: OpenClaw (36) é o "gemini-cli da categoria"; Hermes (35) traz a única implementação fechada de **aprendizado auto-evolutivo** (dimensão 13 promovida a suplementar do template por causa dele); IronClaw (34) redefine o teto conceitual de segurança — o loop estruturalmente incapaz de agir sem o kernel (trust class inforjável por tipos, aprovações como leases por invocação, WASM fail-closed) — algo que **nenhum harness de código avaliado tem**.

O harness embutido confirmou a tese da categoria: as dimensões fracas do n8n são exatamente as que o motor de workflow dispensa (execuções curtas → sem compactação; o plano é o grafo desenhado; permissão é topologia). E a V3 revelou movimento inverso ao esperado: o n8n está *reinternalizando* o loop de execução do LangChain para o próprio engine.

Campeões por dimensão (geral, rodadas 1+2)

Dimensão	Referência atual	Menção
Loop	IronClaw (loop ≠ perímetro de segurança)	opencode (durabilidade), gemini-cli (next-speaker)
Contexto	Aider (repo-map) e opencode (epochs)	Codex (server-driven por modelo), Hermes (3 camadas cache-aware)
Compactação	Codex (remota v2) e Goose (3 técnicas)	IronClaw (circuit-breaker de efetividade)
Ferramentas	Goose (MCP-uniforme) e IronClaw (capabilities tipadas)	n8n (`$fromAI`), Aider (edit formats por eval)
MCP	Codex e OpenClaw (client+server completos)	Goose (in-process)
Permissões/sandbox	**IronClaw** (kernel de autoridade)	Codex (3 camadas de SO), OpenClaw (pairing)
Memória	Hermes (multicamada + FTS5)	gemini-cli (git checkpoint), OpenClaw (Dreaming)
Planejamento	gemini-cli e OpenClaw (goals/task flow)	— dimensão mais fraca da indústria inteira
Subagentes	OpenClaw (push-based + ACP de terceiros)	Codex (graph store), OpenHarness (swarm)
Verificação	gemini-cli (4 suítes) e IronClaw (isolamento cross-tenant)	Aider (benchmark guiando design), Goose (leaderboard)
Extensibilidade	empate amplo — virou commodity	OpenClaw (ClawHub c/ scan), Goose (providers JSON)
Interfaces	OpenClaw (23 canais + voz + apps)	Codex (1 core → CLI/IDE/desktop/cloud)
Aprendizado (13)	**Hermes** (autônomo) e **gemini-cli** (inbox humana) — dois designs nível 3	IronClaw (extração automática)
Proatividade (14)	OpenClaw (heartbeat c/ contexto leve)	IronClaw (routines engine)

Achados transversais da rodada 2

1. **Planejamento é a dimensão mais fraca da indústria**: nenhum harness novo atingiu 3; a média geral da dimensão 8 é a menor do benchmark. Todo mundo tem todo-list; quase ninguém tem plan→approve→execute imposto.
2. **MCP client+server virou o padrão dos maduros**: Codex, OpenClaw, Hermes, OpenHands, n8n e IronClaw expõem-se como servidores — na rodada 1, nenhum dos três fazia isso no core. O harness como *serviço consumível* consolidou em meses.
3. **ACP emergiu como protocolo de orquestração de harnesses**: OpenClaw, OpenHands e Goose orquestram/integram outros harnesses (Claude Code, Codex, Gemini CLI, opencode) via ACP — a predição do cap. 14 sobre "agente-como-serviço" se confirmou por outra via.
4. **A cláusula de expiração ganhou um caso invertido**: o learning loop do Hermes não espera o modelo melhorar — o par modelo+harness escreve o próprio scaffolding (skills). Auto-expansão em vez de expiração.
5. **Segurança tem agora dois paradigmas distintos**: contenção por SO (Codex — o processo não consegue) e arquitetura de autoridade (IronClaw — o loop não alcança). São complementares, e nenhum harness combina os dois ainda.

Próximos passos registrados

- **Reavaliações retroativas**: dimensão 13 nos harnesses da rodada 1 (o `skill-extraction-agent` do gemini-cli é candidato a 2); ohmo como entrada dedicada na categoria pessoal.
- **Fila**: `OpenHands/software-agent-sdk` (o núcleo que faltou), frameworks (LangGraph, CrewAI, Agents SDK — template adaptado), Cline/Roo (IDE), mini-swe-agent (harness mínimo), Crush, smolagents.
- **Evolução metodológica**: do estático ao comportamental — rodar os harnesses em tarefas padronizadas (o Harbor do Goose e o Benchmark Pack do OpenClaw são modelos a estudar).

Aparato do livro

Glossário

As siglas deste livro, **por extenso**, com uma explicação curta e o **contexto** em que aparecem. No corpo dos capítulos, passar o mouse sobre uma sigla mostra o seu significado (`abbr`); aqui está a referência completa. As expansões foram conferidas no próprio texto (Princípio I).

Agentes, protocolos e orquestração

- **MCP — Model Context Protocol.** Protocolo aberto que padroniza como um harness pluga ferramentas, dados e prompts externos ao modelo. *Aparece em:* cap. 06 (MCP) e cap. 17 (Protocolos).
- **ACP — Agent Client Protocol.** Protocolo (origem Zed) para a conversa **agente ↔ editor/cliente**. Não confundir com o *Agent Communication Protocol* da IBM (também "ACP"), encerrado e fundido ao A2A. *Aparece em:* cap. 13 (Interfaces), cap. 17.
- **MRTR — Multi Round-Trip Requests.** Padrão da spec MCP 2026-07-28 que substitui as requisições iniciadas pelo servidor (sampling/elicitation): o servidor responde `input_required` e o cliente retenta com as respostas. *Aparece em:* cap. 06.
- **DCR — Dynamic Client Registration.** Registro dinâmico de clientes OAuth; depreciado na spec MCP 2026-07-28 em favor do CIMD. *Aparece em:* cap. 06.
- **CIMD — Client ID Metadata Documents.** Sucessor do DCR na autorização do MCP: a identidade do cliente vem de um documento de metadados. *Aparece em:* cap. 06.
- **A2A — Agent-to-Agent.** Protocolo de **delegação entre agentes** (origem Google, doado à Linux Foundation). *Aparece em:* cap. 10 (Subagentes), cap. 17.
- **LSP — Language Server Protocol.** Padrão que inspirou os protocolos de agente: separa a "inteligência" (server) da interface (client). *Aparece em:* cap. 11, cap. 12, cap. 14.
- **RPC — Remote Procedure Call.** Chamar um procedimento em outro processo/máquina como se fosse local; base de vários protocolos. *Aparece em:* caps. 05, 06, 10 e 12.
- **MAST — Multi-Agent System Failure Taxonomy.** Taxonomia de modos de falha de sistemas multiagente (do artigo *"Why Do Multi-Agent LLM Systems Fail?"*). *Aparece em:* cap. 10 (Subagentes), bibliografia.
- **RAG — Retrieval-Augmented Generation.** Geração aumentada por recuperação: buscar trechos relevantes e injetá-los no contexto. *Aparece em:* cap. 03 (Contexto), cap. 08.

Modelos e IA

- **IA — Inteligência Artificial** (em inglês, **AI — Artificial Intelligence**). *Aparece em:* todo o livro.
- **LLM — Large Language Model** (modelo de linguagem grande). O modelo que o harness envolve. *Aparece em:* todo o livro.
- **GPT — Generative Pre-trained Transformer.** Família de modelos de linguagem. *Aparece em:* caps. 01, 05 e 09, bibliografia.
- **SWE-bench / SWE-agent — Software Engineering** (benchmark / agente de engenharia de software). *Aparece em:* cap. 11 (Evals).

Ferramentas, interfaces e rede

- **API — Application Programming Interface.** Contrato pelo qual programas se falam. *Aparece em:* todo o livro.
- **SDK — Software Development Kit.** Kit para construir sobre uma plataforma (ex.: o Agent SDK). *Aparece em:* cap. 12 (Extensibilidade), cap. 13.
- **CLI — Command-Line Interface.** Interface de linha de comando. *Aparece em:* cap. 13.
- **TUI — Text (Terminal) User Interface.** Interface de texto interativa no terminal. *Aparece em:* cap. 13.
- **IDE — Integrated Development Environment.** Ambiente integrado de desenvolvimento (ex.: VS Code). *Aparece em:* cap. 13.
- **UI — User Interface** / **UX — User Experience.** Interface e experiência do usuário. *Aparece em:* cap. 13.
- **HCI — Human-Computer Interaction.** Interação humano-computador (campo científico). *Aparece em:* cap. 13.
- **HTTP — HyperText Transfer Protocol.** Protocolo da web. *Aparece em:* caps. 06, 10, 11 e 13.
- **SSE — Server-Sent Events.** Streaming de eventos do servidor para o cliente (usado no chat). *Aparece em:* cap. 13.
- **JSON — JavaScript Object Notation.** Formato de dados dos schemas de ferramentas. *Aparece em:* cap. 05, 06.
- **SO — Sistema Operacional.** *Aparece em:* cap. 07 (Permissões e Sandboxing).
- **CI — Continuous Integration** (integração contínua). *Aparece em:* cap. 11, aparato.
- **DDD — Domain-Driven Design.** Design orientado a domínio; guia o `harness-zero`. *Aparece em:* construção prática.

Editorial, publicação e pesquisa

- **DOI — Digital Object Identifier.** Identificador persistente da obra (Zenodo). *Aparece em:* aparato, capa.
- **ORCID — Open Researcher and Contributor ID.** Identificador do pesquisador (do autor). *Aparece em:* aparato, "Sobre o autor".
- **ISBN — International Standard Book Number.** Identificador padrão de livros. *Aparece em:* Guia Editorial.
- **CC — Creative Commons.** Família de licenças abertas (o conteúdo é CC BY 4.0). *Aparece em:* licença, aparato.
- **MIT.** Licença permissiva de software (nome vem do *Massachusetts Institute of Technology*); cobre o código. *Aparece em:* licença.
- **ICMJE — International Committee of Medical Journal Editors** e **COPE — Committee on Publication Ethics.** Diretrizes de autoria/ética seguidas na divulgação de co-autoria de IA. *Aparece em:* Guia Editorial §6.
- **ICLR — International Conference on Learning Representations.** Conferência científica citada na bibliografia. *Aparece em:* bibliografia.

Apêndice — O estudo: harnesses avaliados

Este apêndice **mostra o trabalho executado**: a lista completa dos harnesses que passaram pelo estudo, com **de onde vieram** (repositório de origem), **a foto exata que foi lida** (fork/commit/snapshot — a materialização da data de corte do método, cap. 01 §6) e o link para a **avaliação completa** de cada um. O instrumento usado em todas as avaliações é o mesmo: o template `HARNESS_EVAL.md` (e `FRAMEWORK_EVAL.md` para frameworks), aplicado por leitura sistemática de código conforme a metodologia do benchmark.

Como ler esta tabela

- **Origem**: o repositório upstream público.
- **Versão/snapshot**: a versão ou o snapshot lido.
- **Fork/commit (data de corte)**: a foto congelada no fork `GHDaru/*` — é o que garante **reprodutibilidade** (qualquer pessoa pode ler o mesmo commit) e materializa a mitigação de obsolescência do método. Os forks são sincronizados pelo script `scripts/sync-forks.ps1`.
- **Avaliação**: o documento completo (metadados, notas por dimensão com evidência de código, diagnóstico e "o que roubar").

Os 16 avaliados

Harness	Categoria	Origem	Versão/snapshot	Fork/commit lido	Avaliado em	Análise
Aider	harnesses de código	github.com/Aider-AI/aider	snapshot 2026-07	fork GHDaru/aider, commit 5dc9490	2026-07-24 (rodada 2)	avaliação
Codex CLI (OpenAI)	harnesses de código	github.com/openai/codex	snapshot 2026-07	fork GHDaru/codex, commit 000d254	2026-07-24 (rodada 2)	avaliação
gemini-cli	harnesses de código	github.com/google-gemini/gemini-cli	snapshot 2026-07 (main)	—	2026-07-24 (rodada 1, exploratória)	avaliação
Goose (Block / AAIF)	harnesses de código	github.com/block/goose	v1.44.0	fork GHDaru/goose, commit 0038bc7	2026-07-24 (rodada 2)	avaliação
opencode	harnesses de código	github.com/anomalyco/opencode	v1.18.4 (V2 em transição, documentada em `CONTEXT.md`)	—	2026-07-24 (rodada 1, exploratória)	avaliação
OpenHands (Agent Canvas)	harnesses de código	github.com/All-Hands-AI/OpenHands	snapshot 2026-07	fork GHDaru/OpenHands, commit 6b04532	2026-07-24 (rodada 2)	avaliação
OpenHarness	harnesses de código	github.com/HKUDS/OpenHarness	v0.1.9	—	2026-07-24 (rodada 1, exploratória)	avaliação
Hermes Agent (Nous Research)	agentes pessoais self-hosted	github.com/NousResearch/hermes-agent	snapshot 2026-07	fork GHDaru/hermes-agent, commit 55ef425	2026-07-24 (rodada 2)	avaliação
IronClaw (NEAR AI)	agentes pessoais self-hosted	github.com/nearai/ironclaw	snapshot 2026-07	fork GHDaru/ironclaw, commit 073ded0	2026-07-24 (rodada 2)	avaliação
ohmo (OpenHarness)	agentes pessoais self-hosted	github.com/HKUDS/OpenHarness (diretório `ohmo/`)	v0.1.9 — avaliação dedicada, complementar à do OpenHarness (rodada 1)	—	2026-07	avaliação
OpenClaw	agentes pessoais self-hosted	github.com/openclaw/openclaw	snapshot 2026-07	fork GHDaru/openclaw, commit 1e15b18b	2026-07-24 (rodada 2)	avaliação
n8n (nó AI Agent)	harnesses embutidos	github.com/n8n-io/n8n	snapshot 2026-07; pacote avaliado: `packages/@n8n/nodes-langchain` v2.32.0 (135 nós de IA)	fork GHDaru/n8n, commit 55e92cc2	2026-07-24 (rodada 2)	avaliação
CrewAI	frameworks	github.com/crewAIInc/crewAI	v1.15.6 — monorepo com 6 pacotes (`crewai`, `crewai-core`, `crewai-tools` ~79 tools, `cli`, `crewai-files`, `devtools`)	fork GHDaru, commit b3aaaab	2026-07	avaliação
LangGraph	frameworks	github.com/langchain-ai/langgraph	langgraph 1.2.9 — monorepo: core (~28k LOC), prebuilt, checkpoint (+postgres/sqlite/conformance), cli, sdk-py; **~63k LOC de testes (2,3× o código)**	fork GHDaru, commit 1e1ca88	2026-07	avaliação
OpenAI Agents SDK	frameworks	github.com/openai/openai-agents-python	v0.18.3	fork GHDaru, commit 5976333	2026-07	avaliação
Software Agent SDK (OpenHands)	frameworks	github.com/OpenHands/software-agent-sdk	v1.37.1	fork GHDaru, commit 99342c4	2026-07	avaliação

Extensão ext-1 (2026-07-31): a primeira promoção Radar → corpus

O corpus cresceu de 16 para **18** pelo caminho que o próprio livro institucionalizou: o Radar diário encontrou os candidatos (varredura de 2026-07-31), o editor aprovou a promoção, os repositórios foram forkados para leitura congelada e o mesmo instrumento (`HARNESS_EVAL.md`) foi aplicado — rodada **ext-1**, sem tocar as fotos das rodadas 1/2. Ambos passam o teste de inclusão do cap. 01 §4 (código aberto + harness de propósito geral + adoção/representatividade): o Grok Build pela abertura de um harness comercial completo; o Pi como **caso deliberadamente atípico** (a lógica de replicação de Yin pedia um contraponto minimalista, e faltava um no corpus).

Harness	Categoria	Origem	Versão/snapshot	Fork/commit lido	Avaliado em	Análise
Grok Build (xAI)	harnesses de código	github.com/xai-org/grok-build	snapshot 2026-07 (aberto em 2026-07-15, Apache 2.0)	fork GHDaru/grok-build, commit dd04f39	2026-07-31 (rodada ext-1)	avaliação
Pi (Earendil Labs)	harnesses de código	github.com/badlogic/pi-mono	snapshot 2026-07-31	fork GHDaru/pi, commit 7846534	2026-07-31 (rodada ext-1)	avaliação

Extensão ext-2 (2026-08-02): a segunda promoção — e uma categoria nova

O corpus cresceu de 18 para **20** pelo mesmo caminho: o Radar confirmou o QM em fonte primária (varredura de 2026-08-02) e a leitura crítica de um artigo de divulgação levou ao Kimi Code, verificado na fonte; o editor aprovou, os repositórios foram forkados e o instrumento aplicado — rodada **ext-2**. O Kimi Code entra pelo mesmo critério do Grok Build (segundo vendor de modelo abrindo um harness completo — o padrão virou tendência, cap. 14). O QM não coube em nenhum arquétipo existente e **inaugura a categoria "agentes organizacionais"**: a unidade de design é a organização (escopos, permissões por audiência, consentimento, auditoria), e o loop do agente é um motor trocável — inclusive **o próprio Pi, avaliado na ext-1, é aqui uma dependência** (`package.json`). A lógica de replicação de Yin pedia exatamente isso: um caso que testasse o limite da taxonomia.

Harness	Categoria	Origem	Versão/snapshot	Fork/commit lido	Avaliado em	Análise
Kimi Code (Moonshot AI)	harnesses de código	github.com/MoonshotAI/kimi-code	CLI 0.31.1 (aberto em ~2026-06, MIT)	fork GHDaru/kimi-code, commit e22479a	2026-08-02 (rodada ext-2)	avaliação
QM (Y Combinator)	agentes organizacionais	github.com/yc-software/qm	snapshot 2026-07-31 (aberto em 2026-07-31, MIT)	fork GHDaru/qm, commit 7f2c916	2026-08-02 (rodada ext-2)	avaliação

Extensão ext-3 (2026-08-02): avaliado e não incluído — o teste de inclusão funcionando

O método também documenta as recusas. O **Traycer** (Traycer AI) foi indicado pelo editor, forkado e avaliado com o instrumento completo (fork GHDaru/traycer, commit `65fc3d7`, MIT) — e **não passou o teste de inclusão** do cap. 01 §4: o repositório aberto (~513 mil linhas) contém clientes, CLI e um protocolo de orquestração notável, mas **nenhuma das quatro peças do harness** — o Host que executa loop, contexto, ferramentas e controle é binário fechado assinado, com nuvem obrigatória (o `AGENTS.md` do próprio repo declara que Host e backends não estão ali). A avaliação completa (18/36) fica como registro: é o caso mais bem documentado do estudo de "open source" como estratégia de distribuição de cliente, e a evidência central do novo Apêndice — A cadeia de suprimentos, para o qual a leitura rendeu o mapa de 18 harnesses orquestrados.

Extensão ext-4 (2026-08-06): o corpus vai a 21 — e a primeira síntese confrontada

O **Prime Agent** (Prime Intellect) chegou por indicação do editor no mesmo dia do anúncio e foi o primeiro candidato a ameaçar uma **Leitura executiva** em vez de acrescentar um adendo: o lançamento afirma que a compactação de contexto "força o modelo a contornar o próprio scaffolding". A leitura do código **manteve a síntese do cap. 04 e registrou uma ressalva** — a compactação não foi eliminada, foi *subordinada ao agente* (ver cap. 04).

O achado estrutural da rodada é outro: **o Prime Agent é construído sobre o Pi** — mesmos quatro pacotes, LICENSE com dupla titularidade (Mario Zechner + Prime Intellect), README creditando o `pi-mono`. É o **quinto** consumidor do Pi registrado no apêndice da cadeia de suprimentos, e fecha um argumento do cap. 12: o sistema de **menor nota do corpus** (Pi, 26/36 — que recusa metade das dimensões por manifesto) foi a base escolhida por um laboratório de fronteira para construir o harness mais radical do estudo.

Harness	Categoria	Origem	Versão/snapshot	Fork/commit lido	Avaliado em	Análise
Prime Agent (Prime Intellect)	harnesses de código	github.com/PrimeIntellect-ai/prime-agent	workspace 0.7.0 (MIT)	fork GHDaru/prime-agent, commit 0e0d233	2026-08-06 (rodada ext-4)	avaliação

Diagnóstico consolidado

Os **resultados por dimensão** (notas 0–3, com evidência) e o diagnóstico comparativo estão no Comparativo dos Harnesses — incluindo o heatmap interativo. Cada avaliação individual traz, além das notas: o **arquétipo observado** do harness, os pontos fortes com caminhos de arquivo, e a seção **"o que roubar"** (padrões que merecem ser levados para outros harnesses).

> **Nota de método** (cap. 01 §6): a seleção seguiu lógica de replicação (Yin) — casos representativos *e* deliberadamente atípicos; a unidade de análise é o código-fonte; as notas seguem a grade fixa do template (feature analysis, DESMET). O placar de expiração das previsões está no Histórico.

> **Consulte também**: implementações de referência além das avaliadas aqui estão catalogadas em Awesome Harness Engineering — Reference Implementations.

Apêndice — A cadeia de suprimentos dos harnesses

> Mapa capturado em **2026-08-02** (rodadas ext-2/ext-3). Como todo estado da arte deste livro, expira: confronte com o Histórico.

Quando este estudo começou, o corpus era uma lista de **concorrentes**: produtos alternativos que resolviam o mesmo problema. As rodadas ext-2 e ext-3 revelaram outra coisa: os harnesses viraram **fornecedores uns dos outros** — dependências de `package.json`, forks vendorizados, subprocessos, sessões alheias retomadas. Este apêndice mostra o trabalho: quem consome quem, por qual mecanismo, com a evidência de cada elo. "Cadeia de suprimentos" aqui é a imagem da manufatura (a fábrica A produz a peça que a fábrica B monta), e também o sentido de segurança do termo: **quem embute, herda os riscos**.

O mapa (evidência por elo)

Cada linha é um elo verificado por leitura de código no commit congelado da avaliação correspondente (paths relativos à raiz de cada repo).

Consumidor	Fornecedor	O que consome	Mecanismo	Evidência
QM	**Pi**	o motor de agente **default** — com **patch de segurança próprio** aplicado pelo consumidor	dependência npm de um fork re-empacotado (`qm-pi-coding-agent-0.82.0-security.2`)	`package.json:58`
QM	Claude Code	motor alternativo	`@anthropic-ai/claude-agent-sdk` + servidor MCP in-process para ponte de tools	`package.json:50`; `src/harness/claude-harness.ts`
QM	Codex CLI	motor alternativo	dependência `@openai/codex`	`package.json:60`
QM	opencode	motor alternativo	`opencode-ai` + plugin/SDK	`package.json:61-62,72`
Kimi Code	**Pi**	a TUI inteira	fork **vendorizado** de `pi-tui`, com agradecimento público	`packages/pi-tui/`; `README.md:122`
software-agent-sdk	Codex CLI, gemini-cli	harnesses inteiros como executores	subprocessos **ACP** orquestrados pelo `ACPAgent`	`openhands/agent_server/conversation_service.py:723`; `event_service.py:873`
Grok Build	Claude Code, Codex, Cursor	as **sessões** dos concorrentes (retomáveis) e seus artefatos de contexto (AGENTS.md/CLAUDE.md/`.cursor`)	leitura dos formatos nativos + session picker	`crates/codegen/xai-grok-pager/src/views/session_picker.rs`
n8n	LangChain	a fundação do nó AI Agent — em processo de **reinternalização** (V3)	dependências `@langchain/*`	`packages/@n8n/nodes-langchain/package.json`
Pi	← terceiros	provedor xAI chega ao Pi **de fora**, por pacote da comunidade	mecanismo de extensão (`pi-xai-oauth`)	radar 2026-08-01
Traycer	Claude Code	motor GUI+TUI: resume/fork, hooks de ciclo de vida, gestão remota de MCP/plugins/skills	SDK + PTY `claude --resume --fork-session` + hooks → CLI `traycer`	`protocol/src/host/agent/tui/unary-schemas.ts:48-80`; `clients/traycer-cli/src/commands/agent-activity-from-hook.ts`
Traycer	Codex CLI	motor GUI+TUI	`codex app-server` (JSON-RPC) + PTY `codex resume`	`protocol/src/host/agent/tui/unary-schemas.ts:70-80`
Traycer	opencode	motor **e substrato da própria inferência** (servidor OpenCode por usuário atrás do backend Traycer)	PTY + spawn de servidor com header de conta	`protocol/src/common/schemas.ts:70-76`; `agent-runtime.ts:839-849`
Traycer	**Pi** (e o fork Oh My Pi)	motores GUI — o fork *sozinho* motivou a versão v6.0 do protocolo	RPC nativo do Pi	`agent-runtime.ts:925-946`; `provider-schemas.ts:80-135`
Prime Agent	**Pi**	**a base inteira** — os quatro pacotes do Pi, com o *tool set* colapsado num único `ipython` e duas camadas novas por cima	fork de tese; LICENSE com dupla titularidade (Mario Zechner + Prime Intellect)	`LICENSE`; `packages/{agent,ai,coding-agent,tui}/package.json`; `README.md`

Consumidor	Fornecedor	O que consome	Mecanismo	Evidência
Traycer	**Hermes**, **Kimi Code**, Cursor, +ACP	motores GUI (8+ providers via ACP: `hermes acp`, `kimi acp`, `grok agent stdio`, `qwen --acp`…)	processos ACP stdio / `@cursor/sdk`	`agent-runtime.ts:851-941`; `protocol/src/host/agent/shared.ts:35-43`

Somam-se os elos de **produção editorial**: o Traycer materializa skills de registries públicos (anthropics/skills, vercel-labs) pinadas por hash num lockfile (`skills-lock.json`) para os agentes que escrevem o próprio repo — o consumo de harness alheio começando antes do produto existir.

O caso extremo: Traycer, o cockpit que é só cadeia

A rodada **ext-3** avaliou o Traycer (18/36) — um produto cuja proposta *inteira* é consumir harnesses alheios: um cockpit multiplayer (~513 mil linhas abertas) que cataloga no próprio contrato de wire a semântica de resume/fork de **18 CLIs/SDKs concorrentes**, com 6 enums de provider congelados por versão de protocolo. Ele **não passou o teste de inclusão** do cap. 01 §4 — as quatro peças do harness não estão no código aberto: o Host que executa loop, contexto e controle é binário fechado assinado, com nuvem obrigatória (`AGENTS.md` do próprio repo confessa; evidência completa na avaliação). O registro fica por dois motivos: é o caso mais bem documentado de **"open source" como estratégia de distribuição de cliente**, e é a prova de que a camada de orquestração — comprar, dirigir e revender o trabalho de outros harnesses — virou produto autônomo.

Três leituras

1. **"De quem ele é feito?" virou pergunta de avaliação.** Um harness já não se descreve só pelo que faz, mas pelos elos que embute. O Pi alimenta hoje **pelo menos cinco sistemas** (QM como motor, Kimi Code como TUI, Traycer como provider, o fork Oh My Pi — e o Prime Agent como base inteira); uma falha, um CVE ou uma mudança de licença nesse único elo propaga pela cadeia inteira, exatamente como na indústria física.
2. **A sessão virou interface de integração.** Três consumidores diferentes (Grok Build, Traycer, QM) tratam a *sessão* de harnesses alheios como artefato retomável — via formato nativo, âncoras de resume/fork versionadas ou re-semeadura por "fita". É um padrão emergente sem padrão: cada um resolve por engenharia reversa do vizinho. Se um formato de intercâmbio de sessões se padronizar (cap. 17), boa parte deste mapa vira código de compatibilidade — a cláusula de expiração aplicada ao próprio apêndice.
3. **O enforcement não viaja pela cadeia.** Quando o QM roda o Pi, as permissões são as do QM (o Pi não as tem); quando o Traycer dirige 18 harnesses, o modo de permissão é **relay** — e a instrução A2A do Traycer chega a mandar os agentes derivados operarem em `full_access` por default. Quem consome um harness herda as capacidades dele, mas **não herda automaticamente os controles** — o elo mais fraco da cadeia define o risco do conjunto.

O contraponto que confirma

Os dois fornecedores mais consumidos do mapa são também os que levam a cadeia de suprimentos *clássica* mais a sério: o Pi pina dependências e faz allowlist de lifecycle scripts (`--ignore-scripts` em tudo); o QM audita o fornecedor a ponto de **remendá-lo** (o patch de segurança da linha 58). A lição fecha o círculo do cap. 07: na era em que o harness do vizinho é sua dependência, a segurança da cadeia de suprimentos deixou de ser tema de npm e virou tema de **arquitetura de agentes**.

> **Consulte também**: as avaliações completas de cada elo estão no Apêndice — O estudo; a leitura editorial da tendência está no Comparativo (rodada ext-2) e no capítulo 14 — Convergências.

Apêndice — Uso do livro (vivo)

> **Estado da arte capturado em 2026-07** · última revisão 2026-07-29 · histórico e registro de expiração

Um livro que ensina **verificação** (cap. 11) e **observabilidade das interfaces** (cap. 13) deveria conseguir responder sobre si mesmo: *como este livro é usado?* Esta página responde — ao vivo.

O que é medido (e o que não é)

Desde a edição 0.49, o site registra **navegação agregada**: qual página foi visitada, quantas vezes, por sessões **anônimas** — e somente depois que o leitor **aceita o aviso de telemetria** (o banner na primeira visita). Nada além disso:

- **Não** coletamos IP, user-agent, nome, email ou qualquer dado pessoal;
- a sessão é um identificador aleatório gerado pelo navegador, apagável pelo próprio leitor (o comando `/limpar` do companion remove tudo da sessão — direito ao esquecimento);
- o painel abaixo consome uma projeção **estritamente agregada** (`total` e contagens por página) — não existe endpoint público com dados individuais.

O painel vivo

(Os números acima existem apenas na versão online — no PDF esta ilha é omitida por definição.)

Para que serve

Este painel é o mesmo insumo que orienta a **cadência do livro vivo** (ADR 0007): capítulos com mais atenção dos leitores têm prioridade na janela trimestral de revisão, e páginas ignoradas levantam a pergunta editorial certa — falta divulgação, ou falta reescrita?

É também uma demonstração em miniatura do que o livro prega: **instrumentar é a metade barata da verificação** — a metade cara é decidir o que fazer com o número. O registro de decisões fica, como sempre, no Histórico.

Apêndice — Grafo do livro (vivo)

> **Estado da arte capturado em 2026-07** · última revisão 2026-07-30 · histórico e registro de expiração

Todo livro técnico é um grafo disfarçado de sequência: capítulos que se citam, sistemas que aparecem em várias dimensões, conceitos que costuram tudo. Esta página torna o grafo explícito — e **interativo**.

Como este grafo é construído (e por que ele nunca desatualiza)

Os nós e as arestas **não são editados à mão nem gerados por um modelo**: são extraídos **deterministicamente do próprio Markdown** a cada build do site, pelo motor de publicação (`publicar/grafo.mjs`). Uma aresta capítulo→harness existe porque aquele capítulo *menciona* aquele sistema no texto — com o peso igual ao número de menções (Princípio I: cada aresta é evidência textual verificável). Como o build roda a **cada mudança publicada do livro**, o grafo acompanha o conteúdo por construção — atualizá-lo não é um processo, é uma propriedade.

- **Nós** (4 tipos): os 18 **capítulos**; os 16 **sistemas do corpus** do estudo; **conceitos**-chave (MCP, A2A, ACP, LSP, RAG, MAST); as 13 **etapas do harness-zero**.
- **Arestas**: capítulo→capítulo (referências cruzadas "cap. NN"), capítulo→sistema (menções), capítulo→conceito (ocorrências), capítulo→etapa (a trilha Mão na massa).

O grafo interativo

(A visualização existe apenas na versão online — no PDF esta ilha é omitida por definição.)

Guia de leitura

- **Hubs**: os nós maiores concentram conexões — espere ver o cap. 02 (Loop) e sistemas avaliados em todas as dimensões como centros gravitacionais.
- **Pontes**: conceitos como MCP conectam grupos que de outro modo ficariam distantes (o cap. 06 ao cap. 17, o corpus aos protocolos) — é a costura do livro visível.
- **A trilha prática**: filtre por "harness-zero" para ver como as etapas amarram os capítulos teóricos à construção (Backward Design em forma de grafo).
- **Clique em qualquer nó** para isolar a vizinhança e navegar direto para a página correspondente.

Apêndice — harness-um: a implementação de referência

> **Estado da arte capturado em 2026-07** · última revisão 2026-07-31 · histórico e registro de expiração

Depois de dezoito capítulos descrevendo o que um harness tem, a pergunta honesta é: *e se juntássemos tudo?* Este apêndice responde com código. O **harness-um** é a implementação de referência do livro — as features dos capítulos 02–13 reunidas num sistema único, pequeno o bastante para ser lido numa tarde e completo o bastante para ser o ponto de partida do seu.

A figura oficial: o núcleo (o agente) envolto pelos 12 segmentos do anel — os capítulos 02–13, um por feature. A identidade visual é a mesma da capa: o harness é o que está *em volta*.

Por que "harness-um" (e não "openharness")

O nome conta a progressão do livro: o **harness-zero** (Mão na massa) constrói uma feature por etapa, do zero; o **harness-um** é o destino — tudo junto e coeso. E há uma razão editorial: "OpenHarness" **já existe** — é um dos 16 sistemas do corpus deste estudo (HKUDS/OpenHarness, port open-source do Claude Code). Batizar a referência do livro com o nome de um sistema que o próprio livro avalia criaria a confusão que o Princípio I existe para evitar.

A linguagem ubíqua

A decisão central do harness-um não é técnica, é **linguística**: o código fala a língua do livro. Cada termo que os capítulos definiram vira um nome de código idêntico — ler o código é reler o sumário. A tradução para o dialeto de cada API de modelo (hoje, Anthropic Messages) acontece numa única borda (`provedores.py`), a **camada anticorrupção**: se o provedor mudar, o domínio nem fica sabendo.

Termo do livro	No código	Capítulo
Loop do agente	`LoopDoAgente.executar()`	02
Turno (com orçamento)	`max_turnos`	02
Montagem de contexto	`MontadorDeContexto` (camadas nomeadas)	03
Compactação	`Compactador` (resumo + cauda intacta)	04
Ferramenta	`Ferramenta`, `@ferramenta`, `CaixaDeFerramentas`	05
MCP	`ClienteMCP` (stateless, spec 2026-07-28)	06
Permissões	`Politica` → `PERMITIR / PERGUNTAR / NEGAR`	07
Memória durável	`Memoria` (`MEMORIA.md`)	08
Sessão	`Sessao` (JSONL, append-only)	08
Plano como artefato	`Plano` (persistido, re-injetado)	09
Subagente	`tarefa()` — contexto limpo, caixa só-leitura	10
Verificação	`Verificador` (pós-mutação, veredito ao modelo)	11
Gancho (hook)	`Gancho` (determinístico, pode vetar)	12
Habilidade (skill)	`Habilidade` (`SKILL.md`, divulgação progressiva)	12
Interface	REPL (`python -m harness_um`)	13
Provedor	`Provedor` → `ProvedorAnthropic`, `ProvedorEco`	02, 11

Como baixar e rodar

O código vive **neste repositório**, ao lado do livro — em `harness-um/`:

```
git clone https://github.com/GHDaru/harness_engineering.git
cd harness_engineering/harness-um
pip install -e .

# sem chave nenhuma (ProvedorEco, offline):
python -m harness_um --eco 'leia @usar ler_arquivo {"caminho": "README.md"}'

# com modelo real (chave SÓ no ambiente):
export ANTHROPIC_API_KEY=...
python -m harness_um     # REPL: /plano /memoria /contexto /sair
```

O `ProvedorEco` merece a nota: ele é determinístico e obedece diretivas `@usar ferramenta {...}` — o suficiente para exercitar o loop inteiro (tool-use, permissões, ganchos, verificação) **sem rede e sem custo**. É por isso que os testes do harness-um rodam no CI do livro a cada push: a referência não pode apodrecer em silêncio.

harness-zero × harness-um

	harness-zero	harness-um
Propósito	**ensinar a construir** (Backward Design)	**mostrar o conjunto pronto**
Forma	13 etapas, cada uma um app completo	1 pacote coeso (`harness_um/`)
Leitura	durante os capítulos	depois do livro
Análogo	caderno de exercícios	gabarito comentado

Expiração

Como tudo neste livro: o harness-um é a foto de **2026-07** — o cliente MCP já nasce na spec 2026-07-28, mas provedores, esquemas e convenções mudam em meses. A cadência do livro vivo (ADR 0007) cobre também este apêndice; o código carrega a mesma cláusula no README.

Bibliografia científica do livro

> Regra editorial: nenhuma referência entra num capítulo sem status ✓ **validada** (ID↔título confirmado por fonte independente). Revisão de 2026-07-29 (spec 050): **todos os itens ⏳ foram verificados por busca web independente** e promovidos a ✓ (com duas correções registradas: o arXiv 2509.18661 é o *Agentic* AutoSurvey; o ISBN 9780226595146 do Norton é da 1ª ed. 2009). ⭐ = âncora do capítulo.

Status geral

Status	Significado
✓	ID↔título confirmado por busca independente nesta sessão
⏳	Citada de memória ou de fonte única; confirmar antes de citar no corpo

Transversal / Fundamentos (caps. 00–01)

- ⭐ ✓ **From Question Answering to Task Completion: A Survey on Agent System and Harness Design** — arXiv 2606.20683. O survey exatamente no recorte do livro; candidata a espinha teórica do cap. 01.
- ✓ **Recursive Agent Harnesses** — arXiv 2606.13643. Achado da validação; avaliar aderência (harnesses compostos — conecta com caps. 10 e 15).
- ✓ **ReAct: Synergizing Reasoning and Acting in Language Models** (Yao et al.) — arXiv 2210.03629. O paper seminal do loop raciocínio+ação.
- ✓ **Li, Xinzhe** *A Review of Prominent Paradigms for LLM-Based Agents: Tool Use, Planning (Including RAG), and Feedback Learning* — COLING 2025, pp. 9760–9779 (aclanthology; arXiv 2406.05804).
- ✓ **Agent Systems with Harness Engineering** (Tang, Peng, Chen et al., RUC/Gaoling) — OpenReview nM5tDHrQsx · PDF + curadoria (maio/2026; **sem versão arXiv**; PDF de 62 pp. lido na íntegra, spec 065). O segundo survey no recorte do livro — e o complemento do âncora acima: taxonomia scaffold-side convergente com a nossa (workflow/memória/skills/multi-agente) mais um terço inteiro que o livro não cobre (**treinamento agêntico**: RL, recompensas, infra de rollout). A tese central citável: harness engineering como "the joint optimization of both components" (modelo⇄scaffold). Ressalvas de rigor: sem seção de limitações, sem metodologia de survey declarada, amostra de sistemas reais n=3 — e permissões, extensibilidade e interfaces (fortes no nosso benchmark) tratadas como direções futuras, não componentes de primeira classe.

História e proveniência (cap. 01 §2–3) — adicionadas na revisão de rigor

- ✓ **ReAct: Synergizing Reasoning and Acting in Language Models** (Yao et al.) — arXiv 2210.03629, ICLR 2023. O loop Pensamento→Ação→Observação; esqueleto de todo harness.
- ✓ **Introducing GitHub Copilot: your AI pair programmer** (GitHub, jun/2021) — github.blog. Marca o "antes": autocomplete pelo Codex, sem loop/ferramentas.
- ✓ **Function calling and other API updates** (OpenAI, jun/2023) — openai.com. O elo modelo→ferramentas.
- ✓ **Introducing the Model Context Protocol** (Anthropic, nov/2024) — anthropic.com.
- ✓ **Announcing the Agent2Agent Protocol (A2A)** (Google, abr/2025) — developers.googleblog.com.
- ✓ **AGENTS.md** — agents.md. O "README para agentes".
- ✓ **BabyAGI** (Yohei Nakajima, abr/2023) — babyagi.org. · **AutoGPT** (Significant Gravitas, mar/2023) — repositório.
- ✓ **Aider** (Paul Gauthier, 2023) — github.com/Aider-AI/aider.
- ✓ **Chain-of-Thought Prompting Elicits Reasoning in Large Language Models** (Wei et al., 2022) — arXiv 2201.11903. · ✓ **Toolformer: Language Models Can Teach Themselves to Use Tools** (Schick et al., Meta, 2023) — arXiv 2302.04761.

Metodologia do estudo (cap. 01 §6) — adicionadas na revisão de rigor

- ✓ **Hassan, A. E. (2008).** *The Road Ahead for Mining Software Repositories.* FoSM/ICSM 2008. Repositórios como dado primário (MSR).

- ✓ **Runeson, P. & Höst, M. (2009).** *Guidelines for Conducting and Reporting Case Study Research in Software Engineering.* Empirical Software Engineering 14(2). Protocolo de estudo de caso em ES.
- ✓ **Kitchenham, B., Linkman, S. & Law, D. (1997).** *DESMET: a methodology for evaluating software engineering methods and tools.* IEE CCEJ. A feature analysis do benchmark.
- ✓ **Sim, S. E., Easterbrook, S. & Holt, R. C. (2003).** *Using Benchmarking to Advance Research.* ICSE 2003. O benchmark como motor científico.
- ✓ **Stol, K.-J., Ralph, P. & Fitzgerald, B. (2016).** *Grounded Theory in Software Engineering Research.* ICSE 2016. Codificação indutiva de artefatos.
- ✓ **Peffers, K. et al. (2007).** *A Design Science Research Methodology for IS Research.* JMIS 24(3). O processo DSRM do harness-zero.
- ✓ **Yin, R. K. (2018)** *Case Study Research and Applications: Design and Methods*, 6ª ed., SAGE (ISBN 9781506336169). · ✓ **Hevner, March, Park & Ram (2004)** *Design Science in Information Systems Research*, MIS Quarterly 28(1), 75–105. · ✓ **Basili, Caldiera & Rombach (1994)** *The Goal Question Metric Approach*, Encyclopedia of Software Engineering, vol. 1, Wiley, 528–532. · ✓ **Hsieh & Shannon (2005)** *Three Approaches to Qualitative Content Analysis*, Qualitative Health Research 15(9), 1277–1288 (DOI 10.1177/1049732305276687). · ✓ **Cook & Campbell (1979)** *Quasi-Experimentation: Design & Analysis Issues for Field Settings*, Houghton Mifflin (ISBN 9780395307908).

Cap. 02 — Loop do Agente

- ✓ ReAct (acima).
- ✓ **LLM-based Agentic Reasoning Frameworks: A Survey** — arXiv 2508.17692.
- ✓ **A Comprehensive Survey on RL-based Agentic Search** — arXiv 2510.16724 (loop treinado, fronteira do capítulo).

Cap. 03 — Entrega de Contexto

- ⭐ ✓ **A Survey of Context Engineering for Large Language Models** — arXiv 2507.13334.
- ✓ **Lost in the Middle: How Language Models Use Long Contexts** (Liu et al.) — arXiv 2307.03172. A base empírica do "posição importa" (justifica tail preservation e prompts em camadas).
- ✓ **Less Context, Better Agents: Efficient Context Engineering for Long-Horizon Tool-Using LLM Agents** — arXiv 2606.10209.

Cap. 04 — Compactação

- ⭐ ✓ **MemGPT: Towards LLMs as Operating Systems** (Packer et al.) — arXiv 2310.08560. A formulação "memória virtual" que antecipou a escada de compactação.
- ✓ **ContextBudget: Budget-Aware Context Management for Long-Horizon Search Agents** — arXiv 2604.01664.
- ✓ **The Missing Memory Hierarchy: Demand Paging for LLM Context Windows** — arXiv 2603.09023.
- ✓ **CompactionRL: Reinforcement Learning with Context Compaction for Long-Horizon Agents** (Li, Hou, Jing, Tang, Dong — Tsinghua/Z.AI) — arXiv 2607.05378 (preprint, 06-jul-2026; **texto integral lido e citações verificadas**, spec 066). A "terceira via" do adendo do capítulo: sumarização aprendida no treino com recompensa de tarefa (+7,0 Pass@1 SWE-bench Verified no GLM-4.5-Air, Tabela 2); valida a tríade limiar+sumário+cauda do capítulo; limitação declarada: train–test mismatch (acoplamento modelo ↔ harness); e o achado pró-harness da Tabela 1 — trocar só o sumarizador move +6,5 pontos.
- ✓ Lost in the Middle (cap. 03) — fundamenta *o que* preservar.

Cap. 05 — Ferramentas

- ✓ **The Evolution of Tool Use in LLM Agents: From Single-Tool Call to Multi-Tool Orchestration** — arXiv 2603.22862.
- ✓ *Tool Learning with Large Language Models: A Survey* (Qu et al.; aceito na Frontiers of Computer Science) — arXiv 2405.17935 (+ repo).
- ✓ **Gorilla: Large Language Model Connected with Massive APIs** (Patil et al., 2023) — arXiv 2305.15334. · ✓ **ToolLLM: Facilitating LLMs to Master 16000+ Real-world APIs** (Qin et al., 2023) — arXiv 2307.16789.

Cap. 06 — MCP

> Atualização (livro vivo, 2026-07): a lacuna registrada nas rodadas anteriores foi **preenchida** — o MCP acumulou um SoK, benchmarks de *tool poisoning* e auditorias empíricas de servidores. O padrão continua sendo *spec de indústria*; a academia entrou pela porta da **segurança**.

- ⭐ ✓ **Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions** (Hou et al.) — arXiv 2503.23278; também ACM TOSEM. O SoK canônico: ciclo de vida do servidor + taxonomia de ameaças por fase.
- ⭐ ✓ **MCPTox: A Benchmark for Tool Poisoning Attack on Real-World MCP Servers** (Wang, Gao et al.) — arXiv 2508.14925. 45 servidores reais / 353 tools; sucesso de até ~73%; modelos mais capazes foram mais suscetíveis.
- ✓ **Model Context Protocol (MCP) at First Glance: Studying the Security and Maintainability of MCP Servers** (Hasan, Li, Fallahzadeh, Rajbahadur, Adams, Hassan) — arXiv 2506.13538. 1.899 servidores auditados: 7,2% com vulns gerais, 5,5% com *tool poisoning*.
- ✓ **MCP Safety Audit: LLMs with the Model Context Protocol Allow Major Security Exploits** (Radosevich & Halloran) — arXiv 2504.03767. Exploits via tools legitimamente registradas; ferramenta MCPSafetyScanner.
- ✓ **A Survey of Agent Interoperability Protocols: MCP, ACP, A2A, and ANP** (Ehtesham, Singh et al.) — arXiv 2505.02279. Escolher o protocolo pelo contexto de confiança (liga ao cap. 17).
- ✓ **Not what you've signed up for: …Indirect Prompt Injection** (Greshake et al.) — arXiv 2302.12173. A base first-principles: conteúdo recuperado é canal de instrução.
- ~ **Threat Modeling and Analysis of Vulnerabilities to Prompt Injection with Tool Poisoning** — arXiv 2603.22489; MDPI *J. Cybersecurity and Privacy* 6(3):84 (2026). STRIDE+DREAD sobre componentes MCP. *(ID e veículo verificados; lista de autores não confirmada por snippet.)*

Fontes da indústria (docs/vendor/praticantes) na linha do Cap. 06 abaixo.

Cap. 07 — Permissões e Sandboxing

- ⭐ ✓ **Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection** (Greshake et al.) — arXiv 2302.12173. O paper que definiu a ameaça.
- ✓ **A Systematic Survey of Security Threats and Defenses in LLM-Based AI Agents: A Layered Attack Surface Framework** — arXiv 2604.23338.
- ✓ **A Survey on Agentic Security: Applications, Threats and Defenses** — arXiv 2510.06445.
- ✓ **Safety and Security Threats of Computer-Using Agents** — arXiv 2505.10924.

Cap. 08 — Memória e Estado

- ⭐ ✓ **MemGPT: Towards LLMs as Operating Systems** (Packer et al.) — arXiv 2310.08560. Contexto como RAM escassa; tiers recall/archival; o agente pagina via tool ("context page faults").
- ⭐ ✓ **Generative Agents: Interactive Simulacra of Human Behavior** (Park et al.) — arXiv 2304.03442; UIST '23. O *memory stream* e o recall por **recência × importância × relevância** + consolidação por reflexão.
- ⭐ ✓ **Cognitive Architectures for Language Agents (CoALA)** (Sumers et al.) — arXiv 2309.02427. A taxonomia episódica/semântica/procedural + working memory (base Tulving).
- ✓ **A Survey on the Memory Mechanism of LLM-based Agents** (Zhang et al.) — arXiv 2404.13501; ACM TOIS. Fontes · formas · operações (escrita/gestão/leitura).
- ✓ **MemoryBank: Enhancing LLMs with Long-Term Memory** (Zhong et al.) — arXiv 2305.10250; AAAI '24. Esquecimento controlado por curva de Ebbinghaus (tempo × frequência de acesso).
- ✓ **Reflexion: Language Agents with Verbal Reinforcement Learning** (Shinn et al.) — arXiv 2303.11366; NeurIPS '23. Auto-reflexão verbal persistida em buffer episódico (ponte com o cap. 16).
- ✓ **A-MEM: Agentic Memory for LLM Agents** (Xu et al.) — arXiv 2502.12110. Notas estruturadas auto-organizadas (Zettelkasten).
- ✓ **Mem0: Production-Ready AI Agents with Scalable Long-Term Memory** (Chhikara et al.) — arXiv 2504.19413; ECAI '25. Pipeline extrair→consolidar→recuperar; benchmark LoCoMo.
- ✓ **A Survey on the Memory Mechanism** e surveys de evolução: **From Storage to Experience** — arXiv 2605.06716; **From Human Memory to AI Memory** — arXiv 2504.15965; **Governing Evolving Memory in LLM Agents (SSGM)** — arXiv 2603.11768 (também cap. 16).

- ~ **Zep: A Temporal Knowledge Graph Architecture for Agent Memory** — arXiv 2501.13956. Grafo bi-temporal; fatos desatualizados invalidados, não deletados. *(ID recorrente em buscas; não aberto byte-a-byte pelo proxy.)*

Cap. 09 — Planejamento

- ⭐ ✓ **ReAct: Synergizing Reasoning and Acting in Language Models** (Yao et al.) — arXiv 2210.03629; ICLR '23. Intercalar razão e ação no mesmo loop.
- ⭐ ✓ **Understanding the Planning of LLM Agents: A Survey** (Huang et al.) — arXiv 2402.02716. Taxonomia de cinco vias (decomposição · seleção · módulo externo · reflexão · memória).
- ✓ **Plan-and-Solve Prompting** (Wang et al.) — arXiv 2305.04091; ACL '23. Plano explícito antes de resolver (escopo conhecido).
- ✓ **Tree of Thoughts** (Yao et al.) — arXiv 2305.10601; NeurIPS '23. Busca sobre planos com backtracking.
- ✓ **ADaPT: As-Needed Decomposition and Planning** (Prasad et al.) — arXiv 2311.05772; NAACL Findings '24. Decompor só quando o executor falha.
- ✓ **Beyond Entangled Planning: Task-Decoupled Planning for Long-Horizon Agents** — arXiv 2601.07577. DAG de sub-objetivos com contexto escopado (−82% tokens).
- ✓ **PlanGenLLMs: A Modern Survey of LLM Planning Capabilities** (Wei et al.) — arXiv 2502.11221; ACL '25. Seis critérios de avaliação de plano.
- ✓ **PLANET: Benchmarks for Evaluating LLMs' Planning Capabilities** — arXiv 2504.14773.
- ✓ **PlanBench** (Valmeekam et al.) — arXiv 2206.10498; NeurIPS '22 Datasets. Modelos crus falham em geração de plano → validadores externos.
- ✓ **TravelPlanner** (Xie et al.) — arXiv 2402.01622; ICML '24. Agentes perdem o fio de múltiplas restrições → externalizar rastreio.

Cap. 10 — Subagentes e Orquestração

- ⭐ ✓ **Why Do Multi-Agent LLM Systems Fail? (MAST)** (Cemri, Pan, Yang et al.) — arXiv 2503.13657. 14 modos de falha em 3 categorias; a maioria vem do *design*, não do modelo — o paper mais acionável para quem constrói orquestração.
- ⭐ ✓ **MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework** (Hong et al.) — arXiv 2308.00352; ICLR '24. SOPs + papéis de linha de montagem contra alucinação em cascata.
- ✓ **AutoGen: Multi-Agent Conversation** (Wu et al.) — arXiv 2308.08155. Agentes "conversáveis" com topologia de interação programável.
- ✓ **CAMEL: Communicative Agents** (Li et al.) — arXiv 2303.17760; NeurIPS '23. Inception-prompting para estabilidade de papel (role-play deriva).
- ✓ **ChatDev: Communicative Agents for Software Development** (Qian et al.) — arXiv 2307.07924; ACL '24. Chat chain + "communicative dehallucination".
- ✓ **AgentVerse** (Chen et al.) — arXiv 2308.10848; ICLR '24. Recrutamento dinâmico + guardrails para comportamento emergente.
- ✓ **LLM-based Multi-Agents: A Survey of Progress and Challenges** (Guo et al.) — arXiv 2402.01680; IJCAI '24. Taxonomia (interface · perfis/papéis · comunicação · capacidade).
- ✓ **Improving Factuality and Reasoning through Multiagent Debate** (Du et al.) — arXiv 2305.14325; ICML '24. Debate como primitiva de verificação.
- ~ **Should We Be Going MAD?** (Smit et al.) — arXiv 2311.17371 · **Stop Overvaluing Multi-Agent Debate** (Zhang et al.) — arXiv 2502.08788. O contrapeso cético: compare com baseline single-agent *compute-matched* antes de adotar a complexidade.
- ✓ **D3MAS: Decompose, Deduce, Distribute** — arXiv 2510.10585.

Cap. 11 — Verificação e Evals

- ⭐ ✓ **SWE-bench: Can Language Models Resolve Real-World GitHub Issues?** (Jimenez et al.) — arXiv 2310.06770; ICLR '24. Grading por execução dos testes reais do repo (FAIL_TO_PASS/PASS_TO_PASS), não string-match.
- ⭐ ✓ **Large Language Models Cannot Self-Correct Reasoning Yet** (Huang et al.) — arXiv 2310.01798; ICLR '24. Não confie na auto-correção *intrínseca* — é preciso verificador externo.
- ✓ **SWE-agent: Agent-Computer Interfaces Enable Automated SE** (Yang et al.) — arXiv 2405.15793; NeurIPS '24. A ergonomia de tools (ACI) dirige o sucesso, não só o modelo.

- ✓ **Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena** (Zheng et al.) — arXiv 2306.05685; NeurIPS '23. Juiz LLM é viável (~80% de acordo) mas tem vieses (posição/verbosidade/self-preference).
- ✓ **A Survey on LLM-as-a-Judge** (Gu et al.) — arXiv 2411.15594. Confiabilidade do juiz como preocupação de pipeline (rubricas, gold set, auto-acordo).
- ✓ **CRITIC: LLMs Can Self-Correct with Tool-Interactive Critiquing** (Gou et al.) — arXiv 2305.11738; ICLR '24. Auto-crítica ancorada em tool (o código roda? o fato confere?) supera introspecção.
- ✓ **Self-Consistency Improves CoT** (Wang et al.) — arXiv 2203.11171; ICLR '23. Amostrar caminhos + voto majoritário: verificação barata só-modelo.
- ✓ **τ-bench: Tool-Agent-User Interaction** (Yao et al.) — arXiv 2406.12045. Verificar o *estado final do mundo* (não o transcript); pass^k revela inconsistência.
- ✓ **Survey on Evaluation of LLM-based Agents** (Yehudai et al.) — arXiv 2503.16416. Eixos: capacidade · segurança · robustez · custo; preferir benchmarks held-out.
- ✓ **Tülu 3 / RLVR** (Lambert et al., Ai2) — arXiv 2411.15124. Reinforcement Learning with Verifiable Rewards: um verificador determinístico é sinal e recompensa mais difícil de fraudar.
- ~ **Reward Hacking in Language Model Agents (AI Safety Gridworlds)** — arXiv 2606.15385; **Do Coding Agents Deceive Us? (Capped Evaluation with Randomized Tests)** — arXiv 2606.07379. O agente joga contra o verificador → held-out/randomizado + testes imutáveis. *(recentes; ID por busca cruzada.)*
- ✓ **The 2025 AI Agent Index** — arXiv 2602.17753 (FAccT '26).
- ✓ **Rethinking the Evaluation of Harness Evolution for Agents** (Wang et al. — AI2/UW/indep.) — arXiv 2607.12227 (preprint, 14-jul-2026; **texto integral lido e citações verificadas**, spec 066 — duas frases que circulavam como citação eram paráfrases de terceiros e foram substituídas pelo verbatim). O paper de método do adendo do capítulo: evolução automática de harness não supera consistentemente test-time scaling sob orçamento equiparado (K=5; Tabelas 1–2), generaliza +0,6 em held-out (Tabela 3), e "most edits memorize fixes rather than distilling strategies" (§5.1) — as três regras (orçamento equiparado, separação busca/avaliação, instrumento sensível a design) valem para qualquer avaliação de harness, inclusive a deste livro.

Cap. 12 — Extensibilidade

> Não há canon acadêmico de *extensibilidade de harness de agente* (lacuna confirmada em 2026-07). As citações duráveis são a SE clássica de arquiteturas extensíveis + a segurança de ecossistemas de plugin.

- ⭐ ✓ **On Plug-ins and Extensible Architectures** (Dorian Birsan) — *ACM Queue* 3(2):40–46 (2005), DOI 10.1145/1053331.1053345. O modelo de plug-in do Eclipse e a advertência do "plug-in hell".
- ⭐ ✓ **LLM Platform Security: …OpenAI's ChatGPT Plugins** (Iqbal, Kohno, Roesner) — arXiv 2309.10254; AIES '24. O "trust triangle" plataforma/plugin/usuário — extensão de terceiros não é confiável por padrão.
- ✓ **Policy/Mechanism Separation in Hydra** (Levin, Cohen, Corwin, Pollack, Wulf) — SOSP '75, DOI 10.1145/800213.806531. A origem de "separar mecanismo de política": o harness dá mecanismo, a extensão dá política.
- ✓ **Protecting Browsers from Extension Vulnerabilities** (Barth, Felt, Saxena, Boodman) — NDSS '10. Over-privilege: 88% das extensões pedem mais poder do que precisam → least-privilege + isolamento.
- ✓ **AIOS: LLM Agent Operating System** (Mei et al.) — arXiv 2403.16971. Kernel que isola escalonamento/memória/tools das aplicações-agente (microkernel aplicado a agentes).
- Fundações SE (livros canônicos): **Microkernel pattern** (Buschmann et al., *POSA* v.1, 1996); **Software Product Lines** (Clements & Northrop, 2001); **Open-Closed Principle** (Meyer, *OOSC*, 1988; Martin, 1996) — "aberto para extensão, fechado para modificação".
- Auto-extensão (ponte com o cap. 16): **Voyager** 2305.16291, **CREATOR** 2305.14318, **CRAFT** 2309.17428, **ToolMaker** 2502.11705.

Cap. 13 — Interfaces

> Não há canon acadêmico de *interface de harness de agente* (lacuna confirmada em 2026-07). As citações duráveis vêm da HCI de interação humano-IA, mixed-initiative e níveis de automação — mais um filete recente (2025-26) de trabalho sobre human-in-the-loop de agentes.

- ⭐ ✓ **Principles of Mixed-Initiative User Interfaces** (Horvitz) — CHI '99, DOI 10.1145/302979.303030. Os 12 princípios de quando o sistema deve agir × perguntar (a decisão de "passar a iniciativa").
- ⭐ ✓ **Guidelines for Human-AI Interaction** (Amershi et al.) — CHI '19, DOI 10.1145/3290605.3300233. 18 diretrizes por fase; a UX de "quando errar" (correção/desfazer barato).
- ✓ **A Model for Types and Levels of Human Interaction with Automation** (Parasuraman, Sheridan, Wickens) — IEEE SMC-A 30(3), 2000, DOI 10.1109/3468.844354. Automação por estágio (aquisição/análise/decisão/ação): o dial de autonomia não precisa ser global.
- ✓ **Human and Computer Control of Undersea Teleoperators** (Sheridan & Verplank) — MIT tech report, 1978. A escala de 10 níveis de automação (o dial de autonomia ajustável).
- ✓ **To Trust or to Think: Cognitive Forcing Functions Can Reduce Overreliance on AI** (Buçinca, Malaya, Gajos) — CSCW '21, arXiv 2102.09692. Explicação sozinha não cura over-reliance; forcing functions sim (a aprovação deve ser ato deliberado).
- ✓ **Overreliance on AI: Literature Review** (Passi & Vorvoreanu) — Microsoft Aether, MSR-TR-2022-12 (2022). Síntese do risco de falsa sensação de supervisão.
- ✓ **Magentic-UI: Towards Human-in-the-loop Agentic Systems** (Mozannar et al., Microsoft) — arXiv 2507.22358. Co-planning/co-tasking e **action guards** = gating de permissão.
- ✓ **Design Considerations for Human Oversight of AI** (Faas et al.) — IUI '26, arXiv 2510.19512. Doze considerações para manter o humano *engajado*, não só presente.
- ~ **LLM-Based Human-Agent Collaboration and Interaction Systems: A Survey** (Zou et al.) — arXiv 2505.00753 (ACL '26). Ponto de entrada de bibliografia. · **Explanation in AI** (Miller) — *Artificial Intelligence* 267 (2019). Explicações contrastivas e seletivas.

Cap. 16 — Aprendizado e Auto-melhoria

- ⭐ ✓ **A Survey of Self-Evolving Agents: What, When, How, and Where to Evolve** — arXiv 2507.21046.
- ✓ **Voyager: An Open-Ended Embodied Agent with LLMs** — arXiv 2305.16291. A skill library auto-escrita que antecipou o Hermes em 3 anos.
- ✓ **Adaptation of Agentic AI: Post-Training, Memory, and Skills** — arXiv 2512.16301.
- ✓ *A Comprehensive Survey of Self-Evolving AI Agents: A New Paradigm Bridging Foundation Models and Lifelong Agentic Systems* (Fang et al., 2025) — arXiv 2508.07407.
- ✓ SSGM (cap. 08) — o risco do aprendizado permanente envenenado.

Caps. 15, 17 — a lacuna registrada

Harnesses embutidos e protocolos têm literatura acadêmica **rarefeita** (buscas de 2026-07 não retornaram surveys dedicados). Registro editorial: o livro cobre essas dimensões com specs, evidência do benchmark e literatura industrial — e assinala a lacuna como oportunidade de pesquisa (possível seção "problemas em aberto" no cap. 14). Nota: os caps. 12 (extensibilidade) e 13 (interfaces), antes nesta lista, foram ancorados em literatura adjacente — SE clássica de arquiteturas extensíveis e HCI de interação humano-IA, respectivamente (ver as seções Cap. 12 e Cap. 13 acima). A lacuna *agent-specific* persiste, mas as fundações duráveis existem.

Coleções vivas

- ✓ **Awesome Harness Engineering** — coleção curada pelo autor: recursos, padrões e templates de harness engineering organizados **por problema** (mesma taxonomia deste livro). Referenciada nos capítulos como "Consulte também", seção a seção.

Fontes da indústria por capítulo (docs de vendor e blogs de engenharia)

> Material comercial/industrial que fundamenta a seção "Fontes da indústria" de cada capítulo (esqueleto v3). URLs verificadas como existentes por busca; fetch direto a anthropic.com/openai.com retorna 403 (anti-bot) neste ambiente — conteúdo confirmado por snippets e citações de terceiros.

Cap. 06 (MCP) — release 2026-07-28: ✓ The 2026-07-28 Specification (blog oficial) · changelog — núcleo stateless, MRTR, extensões, `ttlMs`, política de depreciação. Verificadas por fetch direto do anúncio em 2026-07-31 (spec 060).

Cap. 02 (Loop): How the agent loop works · Loop engineering · Building Effective AI Agents · Running agents (OpenAI Agents SDK) · LoopAgent (Google ADK) · Durable AI Loops (Restate) · Durable Execution (Inngest)

Cap. 03 (Contexto): Effective context engineering · Prompt caching (docs) · Prompt caching is everything · AGENTS.md · Agentic AI Foundation · How Claude remembers your project · AGENTS.md Field Guide 2026

Cap. 04 (Compactação): Compaction (docs) · Auto Compact explained (CometAPI) · Compaction explained (okhlopkov) · Protecting more context (hyperdev)

Cap. 05 (Tools): Writing effective tools for AI agents · Code execution with MCP · Tool search tool (docs) · Advanced tool use · Programmatic tool calling (docs) · Code Mode (Cloudflare) · Apply Patch (OpenAI docs) · GPT-5.1 for developers

Cap. 06 (MCP): Arquitetura MCP (spec) · Transportes (spec) · Introducing MCP (Anthropic) · OpenAI adota MCP (TechCrunch) · Google embraces MCP (The New Stack) · MCP GA no Copilot Studio (Microsoft) · MCP Auth spec (Descope) · Tool Poisoning (Invariant Labs) · Line jumping (Trail of Bits) · The lethal trifecta (Willison) · MCP Registry (preview) · MCP → Agentic AI Foundation (Anthropic)

Cap. 08 (Memória/estado): Manage sessions (Claude Code) · Checkpointing (Claude Code) · File-checkpointing (Agent SDK) · How Claude remembers your project · Memory tool · Managing context (context editing + memory) · Effective harnesses for long-running agents · Memory blocks (Letta) · RAG is not agent memory (Letta) · Memory types (mem0) · Graphiti knowledge-graph memory (Neo4j) · LangMem SDK (LangChain) · Memory vs RAG (AWS Bedrock AgentCore)

Cap. 09 (Planejamento): Permission modes / plan mode (Claude Code) · Best practices — Explore/Plan/Code/Commit · Todo tracking (Agent SDK) · The "think" tool · Extended/interleaved thinking · GitHub Spec Kit · Spec-driven development (GitHub Blog) · Kiro specs · Multi-agent research system (Anthropic) · Don't Build Multi-Agents (Cognition)

Cap. 10 (Subagentes/orquestração): Custom subagents (Claude Code) · Subagents (Agent SDK) · Multi-agent research system (Anthropic) · When to use multi-agent (Claude) · Don't Build Multi-Agents (Cognition) · Agents SDK orchestration (OpenAI) · CrewAI processes · LangGraph multi-agent (LangChain) · Magentic-One (AutoGen) · Multi-agent patterns in ADK (Google) · A2A spec · ACP joins A2A (LF AI & Data)

Cap. 11 (Verificação/evals): SWE-bench Verified (OpenAI) · Why we no longer evaluate SWE-bench Verified · SWE-bench (site) · Terminal-Bench · Define success criteria / build evals (Claude) · Demystifying evals for AI agents (Anthropic) · Statistical approach to model evals (Anthropic) · Effective harnesses for long-running agents · Best practices — TDD · OpenAI Evals · Inspect (UK AISI) · promptfoo · Braintrust scorers · LangSmith LLM-as-judge · Natural emergent misalignment from reward hacking (Anthropic PDF)

Cap. 12 (Extensibilidade): Hooks (Claude Code) · Discover/install plugins · Plugin marketplaces · Customize with plugins (anúncio) · Skills / comandos custom · Settings (precedência/managed) · Advanced tool use (carregamento tardio) · AGENTS.md (padrão aberto) · Codex config · Copilot Extensions (GitHub)

Cap. 13 (Interfaces): Platforms and integrations (Claude Code) · Claude Code on the web · Headless · Agent SDK overview · Managed Agents · VS Code · JetBrains · Copilot agent mode (VS Code) · Permission modes · Streaming output (SDK) · AskUserQuestion / user input (SDK) · Agent Inbox (LangChain) · Channels · Slack

Cap. 07 (Segurança): Claude Code sandboxing · How we contain Claude · Agent approvals & security (Codex) · Agents Rule of Two (Meta) · The lethal trifecta (Willison) · New prompt injection papers · OpenClaw attacks (The Hacker News)

Pedagogia (fundamenta o método do livro, não o conteúdo)

- ✓ **Blueprints for complex learning: The 4C/ID-model** (van Merriënboer et al.) — ETR&D.
- ✓ **Cognitive Architecture and Instructional Design: 20 Years Later** (Sweller, van Merriënboer & Paas, 2019) — EPR.
- ✓ **van Merriënboer & Kirschner (2018)** *Ten Steps to Complex Learning*, 3ª ed., Routledge (ISBN 9781138080805). · ✓ **Wiggins & McTighe (2005)** *Understanding by Design*, expanded 2nd ed., ASCD (ISBN 9781416600350). · ✓ **Diátaxis** (Procida) — diataxis.fr.

Guia — Metodologias de escrita (survey do Guia Editorial §6)

> Fontes do estudo sobre processos/metodologias de escrita editorial e acadêmica (Guia Editorial §6). Todas verificadas por busca cruzada (revisão spec 050). Feature `010-estudo-metodologias-escrita`.

Tradicionais:

- ✓ **The IMRAD Structure: A Fifty-Year Survey** (Sollaci & Pereira, 2004) — *J. Med. Libr. Assoc.* 92(3):364–371, PMC442179.
- ✓ **The Science of Scientific Writing** (Gopen & Swan, 1990) — *American Scientist* 78(6):550–558, JSTOR 29774235.
- ✓ **How to Write and Publish a Scientific Paper** (Day & Gastel) — 7ª ed. Cambridge, ISBN 9781107670747.
- ✓ **A Cognitive Process Theory of Writing** (Flower & Hayes, 1981) — *CCC* 32(4):365–387, DOI 10.58680/ccc198115885.
- ✓ **Revision Strategies of Student Writers and Experienced Adult Writers** (Sommers, 1980) — *CCC* 31(4):378–388, DOI 10.2307/356588.
- ✓ **The Elements of Style** (Strunk & White, 4ª ed. 2000) — ISBN 9780205309023 · **Style: Toward Clarity and Grace** (Williams, 1990) — ISBN 9780226899152 · **On Writing Well** (Zinsser, 2006) — ISBN 9780060891541.
- ✓ **The Chicago Manual of Style** (17ª ed., 2017) — ISBN 9780226287058 · **APA Publication Manual** (7ª ed., 2020) — ISBN 9781433832161.
- ✓ **The Craft of Research** (Booth, Colomb, Williams et al., 4ª ed. 2016) — ISBN 9780226239736 · **The Uses of Argument** (Toulmin, 1958) — Cambridge University Press.
- ✓ **The history of the peer-review process** (Spier, 2002) — *Trends in Biotechnology* 20(8):357–358, DOI 10.1016/S0167-7799(02)01985-6.
- ✓ **Peer Review** (Melinda Baldwin) — Encyclopedia of the History of Science (CMU ETHOS, ed. Christopher Phillips), entrada *(entrada sem ano declarado)*. · ✓ **Developmental Editing** (Scott Norton) — Univ. of Chicago Press, 1ª ed. 2009, ISBN 9780226595146 *(há 2ª ed. 2023, ISBN 9780226793634)*.
- (Pedagogia — ver seção acima: Backward Design; 4C/ID; Sweller; Diátaxis.)

Era-IA:

- ✓ **CoAuthor** (Lee, Liang, Yang, 2022) — CHI '22, DOI 10.1145/3491102.3502030; arXiv 2201.06796.
- ✓ **Wordcraft** (Yuan, Coenen, Reif, Ippolito, 2022) — IUI '22, DOI 10.1145/3490099.3511105; arXiv 2107.07430.
- ✓ **Co-Writing with Opinionated Language Models Affects Users' Views** (Jakesch et al., 2023) — CHI '23, DOI 10.1145/3544548.3581196; arXiv 2302.00560.
- ✓ **Spec Kit** (github.com/github/spec-kit); **Kiro** (kiro.dev); **Structured Authoring in Docs-as-Code** (SIGDOC '24) — DOI 10.1145/3641237.3691677; **DITA** (dita-lang.org).
- ✓ **RAG** (Lewis et al., 2020) — arXiv 2005.11401.
- ✓ **RARR** (Gao et al., 2023) — ACL '23, arXiv 2210.08726 · **Evaluating Verifiability in Generative Search Engines** (Liu, Zhang, Liang, 2023) — arXiv 2304.09848 · **A Watermark for LLMs** (Kirchenbauer et al., 2023) — arXiv 2301.10226.
- ✓ **ICMJE** (AI use by authors); **COPE — Authorship and AI Tools** (2023) (position); **Thorp, "ChatGPT is fun, but not an author"** (*Science*, 2023) — DOI 10.1126/science.adg7879; **Nature editorial** (2023) — d41586-023-00191-1.
- ✓ **Fabrication and errors in the bibliographic citations generated by ChatGPT** (Walters & Wilder, 2023) — *Scientific Reports* 13, DOI 10.1038/s41598-023-41032-5.
- ✓ **Your Brain on ChatGPT** (Kosmyna et al., 2025) — arXiv 2506.08872 · **Homogenization Effects of LLMs on Human Creative Ideation** (2024) — arXiv 2402.01536 · **Academ-AI** (2024) — arXiv 2411.15218.
- ✓ *Agentic AutoSurvey: Let LLMs Survey LLMs* (Liu et al., 2025) — arXiv 2509.18661 *(nota: é o "Agentic AutoSurvey"; o AutoSurvey original é trabalho anterior distinto)*. · ✓ **Defeating Nondeterminism in LLM Inference** (Thinking Machines Lab, set/2025) — blog de indústria, não-acadêmico.

Histórico — este é um livro vivo

> A engenharia de harness muda em meses. Este livro assume isso: cada capítulo declara **quando** seu estado da arte foi capturado, e este arquivo registra o que mudou entre edições. É a materialização da tese central do livro — a **cláusula de expiração** (cap. 01, 14): todo componente de harness é temporário; um livro sobre isso precisa ser datado, ou contradiz o que ensina.

Como ler as datas do livro

- **Data do evento** (no corpo dos capítulos): quando algo aconteceu no mundo — "AGENTS.md doado à Linux Foundation (dez/2025)". É fato histórico, não muda.
- **Data de captura / "estado da arte em"** (no cabeçalho de cada capítulo): quando *nós* fotografamos o panorama. É o que diz ao leitor se a seção "Estado da arte" está fresca. Uma seção capturada em 2026-07 lida em 2028 deve ser confrontada com este histórico.
- **Rodada do benchmark** (nas avaliações): a versão da foto de cada repositório (`rodada 1`, `rodada 2`, `frameworks-1`), com data. Reavaliar = nova rodada, nunca sobrescrever silenciosamente.

Tabela de snapshot por capítulo

Capítulo	Estado da arte capturado em	Fontes da indústria	Última revisão
02 Loop	2026-07	✓	2026-07-25
03 Contexto	2026-07	✓	2026-07-25
04 Compactação	2026-07	✓	2026-07-25
05 Ferramentas	2026-07	✓	2026-07-25
06 MCP	2026-07	✓	2026-07-26
07 Permissões/Segurança	2026-07	✓	2026-07-25
08 Memória e Estado	2026-07	✓	2026-07-26
09 Planejamento	2026-07	✓	2026-07-26
10 Subagentes/Orquestração	2026-07	✓	2026-07-26
11 Verificação/Evals	2026-07	✓	2026-07-26
12 Extensibilidade	2026-07	✓	2026-07-26
13 Interfaces	2026-07	✓	2026-07-26
14 Convergências	2026-07	—	2026-07-28
15 Harness Embutido	2026-07	—	2026-07-28
16 Auto-melhoria	2026-07	—	2026-07-28
17 Protocolos	2026-07	—	2026-07-28
00 Introdução · 01 Fundamentos	2026-07	✓ (01)	2026-07-28

Edições

Edição 0.77 — 2026-08-08 · mostrar o progresso é o convite (spec 093 · ADR 0010)

- **O pedido do editor tinha três partes, e a do meio movia uma fronteira**: o convite para guardar progresso estava discreto demais; ele queria **guardar os e-mails** para avisar sobre livros futuros, *"desde que eles aceitem"*; e o leitor deveria **ver o próprio progresso**. O item do meio colidia com o que já estava escrito em três lugares do site — *"sem informativo"* — e com o e-mail que o leitor recebia. A

ADR 0009 tinha registrado isso como fronteira e exigido nova ADR para movê-la; a ADR 0010 é essa ADR, e o que ela decide é **como** mover sem quebrar promessa.

- **O diagnóstico de UX inverteu o pedido**: o problema do convite não era tamanho, era **momento** — ele aparecia antes de o progresso existir, pedindo para guardar algo que ainda valia zero. E havia um vazio maior: **o leitor não via o próprio progresso em lugar nenhum**. O cartão "Retomar" dizia onde ele parou, nunca quanto tinha andado. A tese que saiu daí: **mostrar o progresso É o convite** — o pedido 3 resolve o pedido 1. O "Retomar" foi substituído por "Sua leitura", com barra e contagem, e a oferta de guardar virou a última linha do cartão, não um banner.
- **Onde eu estava errado, e o dado que me corrigiu**: propus o convite de fim de capítulo a partir do **terceiro**, pela regra genérica de produto ("não peça antes de entregar valor"). O editor perguntou *"por que não antes?"* e a resposta estava na telemetria do próprio livro, que este mesmo Radar já tinha lido: capa **17**, sumário **9**, introdução **7**, capítulo 01 — **3**. Um convite no terceiro capítulo seria visto por ninguém, e quem lê um capítulo e some é **exatamente** quem perde o progresso. Passou para o **primeiro**; o que se adapta é o tom, não o gatilho.
- **Dois consentimentos, nunca embutidos um no outro**: continuidade (o e-mail como chave de progresso) é registrada no `/entrar`, não no `/assinar` — é ali que a posse do endereço fica provada, e gravar no pedido criaria linha em nome de quem só teve o e-mail digitado por um terceiro. Contato (avisar sobre livro novo) é perguntado **depois**, à parte, **desmarcado**, e sair sem responder não grava nada: silêncio vale não, e nunca vale sim.
- **Ninguém foi migrado.** Quem assinou sob "sem informativo" não entrou na lista de contato por efeito da ADR. O editor decidiu **zerar a base** antes de o acordo novo valer — como não há leitores além dele, a migração deixou de existir como problema, e o precedente fica registrado para quando a base não for mais vazia.
- **Consentimento é registro, não interruptor**: `consentimentos` é **append-only** — cada "dei" e cada "revoguei" é uma linha, e o estado atual é a última por (e-mail, finalidade). Um booleano diria o estado e perderia a história, e é a história (quando, e a que texto) que a LGPD pede como prova. **O direito ao esquecimento ganha do append-only**: apagar o leitor apaga também seus consentimentos, porque guardar a prova do consentimento de quem pediu para sumir seria guardar justamente o e-mail que ele mandou apagar.
- **Contar capítulo lido não pediu tabela nova**: `nav_events` já registrava slug × sessão e já seguia o leitor na fusão da spec 080. O backend devolve os slugs visitados; quem sabe o que é capítulo é o site, que tem o sumário. Anônimo funciona inteiro pelo `localStorage` — pedir e-mail para mostrar progresso seria cobrar antes de entregar, o erro que a spec diagnosticou.
- **A rota de exportação nasceu desligada**: `GET /leitores` exige `ADMIN_TOKEN`, que não existe no ambiente. É o mesmo default seguro de `/suggestions` e da telemetria de administração — o editor liga quando for usar a lista.
- **Verificação**: 62 testes de backend (25 novos: dar/revogar/regravar, revogar contato preservando continuidade, descadastro idempotente que não enumera, exportação só com token) e **51 asserções em navegador nos dois idiomas**, percorrendo o caminho real com uma caixa de correio falsa no lugar do provedor — assinar pelo cartão, abrir o link do e-mail, entrar, ver a caixa desmarcada, marcar, aparecer na lista, sair por um clique e confirmar que a continuidade sobreviveu.
- **IA (A3)**: agente **Claude Code (Anthropic)**; ADR, spec, implementação, testes e verificação em navegador.

Edição 0.76 — 2026-08-07 · o endereço do site vira variável (spec 089)

- **Origem**: o editor tem domínio próprio (`ghdaru.com.br`) e decidiu publicar o livro em **`harness.ghdaru.com.br`**, com o repositório fechando. O endereço estava **gravado em código** em três lugares e espalhado por um quarto (o conteúdo) — trocar de hospedagem exigia caçar todos, e o mais perigoso falharia em silêncio.
- **A decisão de não preservar os links antigos foi tomada com dado, não por gosto**: a telemetria pública do próprio livro mostrava **47 visitas** e 11 páginas distintas. Não havia base de leitores a proteger, e manter dois repositórios pelo resto da vida do projeto custaria mais do que vale. Registrado com a ressalva de que o número conta só quem aceitou o aviso de privacidade — o real é maior, mas não em ordens de grandeza.
- **Um endereço, uma variável**: `SITE_URL` passa a controlar canonical, hreflang, og:image e o rodapé dos PDFs — **428 ocorrências no HTML, todas derivadas da mesma origem**. Barra final normalizada, porque esquecê-la produziria `https://exemplo.comindex.html`.
- **O CORS entrou antes do domínio existir**, de propósito. É a falha mais traiçoeira desta migração: o site abriria, o texto apareceria, e **só o companion morreria** — chat, consentimento, telemetria e link mágico, todos em silêncio, sem erro na página. Pré-autorizar custa nada e elimina o passo que seria esquecido justamente no dia da virada.
- **O que a verificação encontrou**: construir com o domínio novo revelou **dois links absolutos escritos à mão dentro do conteúdo** — nenhuma variável os alcança, são texto. Viraram relativos, que é o que sempre deveriam ter sido.
- **E a correção introduziu uma regressão, pega pelo checklist**: trocar por `en/` fez o reescritor não reconhecer a página e cair no repositório, subindo os alvos de repo de **9 para 10**. O alvo certo era `en/index.html`. É a terceira vez que o item 6 do checklist de

verificação pega algo que o build verde deixaria passar.
- **IA (A3)**: agente **Claude Code (Anthropic)**.

Edição 0.75 — 2026-08-07 · o e-mail sai por API HTTP, não por SMTP (spec 087)

- **O que a medição fechou**: com `SMTP_PORT=587` e STARTTLS — o transporte **correto** — o envio ainda morria com `motivo: "conexao"` depois do timeout de socket **inteiro** (20 s). Isso é assinatura de pacote descartado em silêncio por firewall; host inexistente falharia em menos de um segundo. **O egresso SMTP do PaaS é bloqueado**, política comum para conter spam. A senha de app do editor sequer chegou a ser testada.
- **A correção**: o e-mail passa a sair por **API HTTP** (Resend), o mesmo caminho de rede que o backend já usa para o LLM e que comprovadamente funciona nesta infra. Sem dependência nova — `httpx` já era o cliente do LLM.
- **Uma porta única de saída**: `_enviar_email(para, assunto, corpo) -> (ok, motivo)` passa a servir o link mágico **e** as sugestões dos leitores. Quem chama não sabe qual é o transporte — sabe se foi e, se não foi, por quê. O SMTP fica como alternativa escolhida sozinha quando não há chave de API: não é código morto, é portabilidade para quem rodar numa VPS.
- **Descoberta de passagem**: as sugestões dos leitores (E05) nunca chegaram ao autor, pelo mesmo bloqueio. Passava despercebido porque ali o envio é best-effort — a sugestão sempre esteve salva no banco. Falha silenciosa cobra juros.
- **O arco de quatro specs que isto fechou**: 080 acertou em não mentir "enviado", mas engolia o porquê; 084 fez o backend dizer **em que classe** falhou; 085 mostrou **quais variáveis o processo enxerga** — foi o que revelou que estavam noutra infraestrutura; 086 separou "porta errada para o protocolo" de "egresso bloqueado"; 087 troca o transporte. Cada camada existiu porque a anterior não bastou, e nenhuma delas foi palpite.
- **A lição, cara**: eu diagnostiquei errado uma vez, inferindo pelo tempo de resposta que o SMTP "estava tentando". Eram idas ao Neon — cada conexão custa ~2 s. Comparei contra um baseline que não tocava banco. O `EMAIL.md` hoje carrega o aviso explícito de **não** diagnosticar pelo cronômetro.
- **IA (A3)**: agente **Claude Code (Anthropic)**.

Edição 0.74 — 2026-08-07 · o envio do link mágico para de falhar em silêncio (spec 084)

- **Defeito encontrado no primeiro uso real da 080**: com o SMTP já configurado, `POST /assinar` respondia `{"enviado": false}` e mais nada. O `except Exception: return False` transformava credencial recusada, porta bloqueada e TLS na **mesma** resposta — o editor ficava com um botão que não funciona e sem pista, e eu ficava adivinhando entre cinco hipóteses.
- **O que provou que ele estava tentando**: o tempo. `GET /health` responde em 0,34 s; `POST /assinar` respondia em **7,4 s de forma consistente**. `SMTP_HOST` vazio retornaria na primeira linha, sem rede. Medição, não suposição — Princípio I aplicado a um defeito de infra.
- **Correção**: a exceção vai para o `stderr` (log do Railway, do operador) com tipo e mensagem; ao cliente volta só uma **classe grosseira** — `auth`, `conexao`, `tls`, `destinatario`, `smtp`, `desligado`, `outro`. `GET /health` passa a declarar `"smtp"`. O widget traduz a classe para linguagem de leitor, sem jargão de servidor. O token continua fora de log, de resposta e de artefato.
- **A sutileza que virou teste**: em Python 3 `smtplib.SMTPException` **herda de `OSError`**, então um `isinstance(exc, OSError)` no topo do mapa classificaria tudo como `conexao`. A ordem é a lógica, e 13 casos a fixam.
- **A lição, que já estava escrita**: o anti-checklist de `.specify/memory/checklist-verificacao.md` diz para não declarar verificado o que só foi construído. A 080 acertou em não mentir "enviado"; errou em não dizer o porquê. Falha silenciosa é dívida que só vence no dia do uso.
- **IA (A3)**: agente **Claude Code (Anthropic)**.

Edição 0.73 — 2026-08-06 · o e-mail como chave de continuidade, não como muro (spec 080)

- **O defeito**: o leitor era anônimo **por navegador**. `cmp_sid` e `hz_ultimo` viviam só no `localStorage`, então quem lia no notebook e retomava no celular perdia tudo — progresso, objetivo declarado e a conversa inteira com o tutor. Num livro de 24 capítulos lido em sessões separadas por dias, perder o fio era o modo de falha mais provável.
- **A solução, e o que ela deliberadamente não é**: um e-mail vira chave de continuidade — sem senha, sem cadastro, sem área restrita, sem informativo. O leitor informa o e-mail, recebe um **link mágico** de uso único (30 min, guardado só como hash SHA-256) e, ao abri-lo, o navegador **adota a sessão canônica** daquele leitor. **A navegação anônima segue completa**: nenhuma página, download ou função exige e-mail, e o convite é uma linha dispensável no rodapé do painel e sob o cartão "Retomar".
- **O ponto de alavanca**: histórico, objetivo e consentimento já eram indexados por `session_id`. A feature inteira se reduziu a **trocar o `session_id` do navegador pelo canônico do leitor** — o resto seguiu de graça. Só o progresso de leitura, que nunca saía do `localStorage`, ganhou tabela.

- **Fundir, não descartar**: quem conversa antes de assinar não perde a conversa. O link carrega a sessão anônima de quem o pediu, e `/entrar` funde mensagens, objetivo, consentimento, navegação e progresso na canônica — recusando fundir a sessão de **outro** leitor, o que virou teste.
- **Decisões de segurança que valem registro**: a sessão canônica passa a ser gerada no **servidor** com entropia criptográfica (era `crypto.randomUUID()` do navegador — e este id sempre foi, de fato, uma credencial); a resposta de `/assinar` é idêntica para e-mail novo e já cadastrado (sem enumeração); o token sai da URL por `history.replaceState` assim que o POST parte; e sem SMTP a assinatura **falha visivelmente** em vez de mentir "enviado".
- **Verificação**: 32 testes de backend (uso único, expiração, ausência de enumeração, fusão, direito ao esquecimento) e 34 asserções em navegador **nos dois idiomas**, incluindo a promessa central — o capítulo lido no navegador A aparece no cartão "Retomar" do navegador B.
- **Pendente do editor**: `SMTP_HOST/PORT/USER/PASS` no Railway (ver `chat-companion/backend/EMAIL.md`). Até lá o código sobe e todo o resto funciona; o convite diz a verdade sobre o envio desligado.
- **IA (A3)**: agente **Claude Code (Anthropic)**; implementação, testes e verificação em navegador.

Edição 0.72 — 2026-08-06 · o site deixa de depender do repositório (spec 083)

- **Origem**: a verificação da ext-4 encontrou um link quebrado; a investigação mostrou **74 links** do site apontando para o repositório. Com a decisão editorial de **tornar o repositório privado**, isso deixava de ser incômodo e virava bloqueio: as **21 avaliações do benchmark — o ativo central do estudo — ficariam inacessíveis**.
- **Correção**: o motor passa a publicar como páginas do site tudo o que é conteúdo e vivia só no repo — **41 páginas**: as 21 avaliações, a metodologia e os templates do benchmark, a mesa e o contrato do Radar, as 9 ADRs e os 6 estudos de apoio. Resultado: **74 → 7 links** para o repositório.
- **Dois defeitos meus, encontrados na própria implementação**: o mapa de páginas extras existia só na passada PT, então os links EN continuavam indo ao GitHub (as páginas são PT-only, mas o EN precisa linká-las com `../`); e o `jornal.mjs` tinha os links do Radar **em código**, fora do alcance do reescritor.
- **Fica registrado o que não foi resolvido**: os 7 links restantes apontam para **código** (`scripts/`, `harness-um/`, `chat-companion/`, `publicar/`). Com o repo privado, são becos sem saída — decisão do editor entre publicá-los, espelhar o código num repo público ou reescrever as menções.
- **IA (A3)**: agente **Claude Code (Anthropic)**; correção com verificação em navegador nas duas línguas.

Edição 0.71 — 2026-08-06 · rodada ext-4: a primeira síntese confrontada (spec 082)

- **Feature spec-kit oficial `082-ext4-prime-agent`.** O **Prime Agent** (Prime Intellect, MIT, **31/36**) chegou por indicação do editor no dia do anúncio e foi o **primeiro candidato a ameaçar uma Leitura executiva** em vez de acrescentar um adendo. Corpus: 20 → **21**.
- **O veredito do cap. 04: mantido, com ressalva.** O anúncio acusa: *"context compaction força o modelo a contornar o próprio scaffolding"*. O código diz outra coisa — as 1.398 linhas de `core/compaction/` seguem lá, **melhoradas**; o que mudou é **quem manda**: `compact.run()` virou chamável pelo agente, com handler que **agenda em vez de executar** (executar abortaria a célula que pediu), rodando mesmo com a compactação automática desligada, sob 12 testes. A escada de agressividade não ficou obsoleta: **a autoridade sobre ela migrou para o agente**. O capítulo ganhou a seção "A terceira fronteira" e a Leitura executiva ganhou ressalva datada, com a regra explícita: se o padrão se repetir noutros harnesses, o parágrafo será **reescrito, não emendado**.
- **O Pi virou substrato do corpus.** O Prime Agent é o **Pi por baixo** — mesmos quatro pacotes, LICENSE com dupla titularidade (Mario Zechner + Prime Intellect), README creditando o `pi-mono`. É o **quinto** consumidor do Pi no apêndice da cadeia de suprimentos. A ironia fecha o argumento do cap. 12: o sistema de **menor nota do corpus** (Pi, 26/36 — que recusa metade das dimensões por manifesto) foi a base escolhida por um laboratório de fronteira para o harness mais radical do estudo.
- **Novo teto na dimensão 13, novo piso na 10.** O *Continual Harness* dá ao agente **CRUD do próprio estado** (prompt, memória, skill, spec de subagente) com review gate e rollback — supera o Hermes. Mas o repositório **não contém nenhum eval**: o `packages/evals` do Pi desapareceu no fork, e a alegação de 95,5% no ARC-AGI-3 não tem artefato reprodutível no código. Um harness que se auto-modifica sem bancada de medição é a combinação mais arriscada já registrada no benchmark — e ficou dito.
- **Delta traduzido no mesmo ciclo**: 6 arquivos EN (00, 01, cap. 04, apêndice do estudo, cadeia de suprimentos, comparativo), selos renovados.
- **IA (A3)**: agente **Claude Code (Anthropic)** — avaliador sobre o clone congelado com a pergunta "o que ele acrescenta ao Pi?"; curadoria e decisão humanas.

Edição 0.70 — 2026-08-06 · o contrato do Radar aprende com o campo (spec 081)

- **Origem incomum**: as duas mudanças foram **sugeridas pelo próprio agente do Radar**, em execuções seguidas, e ficaram esperando — ele detectou os padrões mas não podia mudar o contrato (regra dura: escrita só em `radar/`, e `AGENTE.md` *é* o processo). O editor aprovou; a spec implementou. É o ciclo do livro vivo funcionando na camada do método, não do conteúdo.
- **O que o campo mostrou**: (1) **três execuções seguidas** com fato real e **data errada na fonte secundária** — o episódio Anthropic×terceiros (de jan–mar) vendido como notícia de agosto, o GA do framework da Microsoft confundido com o GA do harness; (2) um repositório com **194.982 estrelas** — mais que o maior do corpus — que teria entrado como candidato prioritário, e cuja API revelou **109.281 forks** (razão 1,8:1, quando projetos reais ficam em 10:1) e uma autodescrição de *"agent-managed museum exhibit… no human intervention"*, enquanto um press release pago o vendia como framework de produção.
- **Três regras duras novas**: **agregador é pista, nunca fonte** (afirmar exige primária; sem ela, ⏳ dizendo qual falta); **a data merece verificação separada do fato**; e **candidato ao corpus passa pela API do repositório** antes de qualquer recomendação — licença, criação, último push e **estrelas × forks**, com razão abaixo de ~5:1 exigindo desconfiança explícita, mais a leitura da descrição que o repositório dá de si mesmo.
- O caso concreto ficou **citado dentro do contrato**: regra sem história não é lembrada.
- **IA (A3)**: agente **Claude Code (Anthropic)** — o agente do Radar propôs, este ciclo implementou; curadoria e aprovação humanas.

Correção 2026-08-05 · o tour falava do chat com o chat fechado (spec 079)

- **Defeito relatado pelo editor**: no passo do companion, o cartão do tour ficava "perdido na tela" — porque o chat não abria.
- **Dois problemas encadeados**: o passo *Companion* mirava a **bolha do canto** e descrevia o que só existe com o painel aberto ("digite /", "passe o mouse nos chips"); e o passo *Bastidores* mirava `.cmp-status`, que **só existe dentro do painel** — com o chat fechado ele era filtrado e o tour rodava com **4 passos em vez de 5**, perdendo em silêncio justamente o passo que demonstra a tese do livro (tokens, chamadas, contexto injetado).
- **Correção**: passos que falam do painel ganharam a marca `abrir` — o tour **abre o chat** ao chegar neles e remede as posições depois da animação. Verificado em navegador percorrendo o tour a partir do clique real no banner: 5 passos, painel aberto do 3 em diante, spotlight no lugar certo.
- **Registrado de passagem**: o convite do tour só existe no fluxo do banner de consentimento — quem já aceitou nunca o recebe, só chega por `/tour`. Decisão implícita, agora explícita, para o editor confirmar ou mudar.
- **IA (A3)**: agente **Claude Code (Anthropic)**; correção (Princípio VII) com decisão humana explícita.

Correção 2026-08-05 · a telemetria estava zerada — e era permanente (spec 078)

- **Defeito relatado pelo editor**: o Apêndice — Uso do livro e o contador do rodapé marcavam **zero**. O backend estava de pé e o caminho funcionava (testado ao vivo: `/consent` seguido de `/telemetry` grava e o agregado sobe).
- **Causa raiz — contrato quebrado entre cliente e servidor**: o aceite vivia em dois lugares. O cliente gravava o flag no `localStorage` e, com ele presente, **nunca mais mostrava o banner**; o servidor exigia a linha de consentimento daquela sessão e, sem ela, descartava a navegação devolvendo `{"ok": false}`. E o POST do aceite era `fetch(...).catch(function(){})` — **falha em silêncio**. Bastava o backend estar hibernando, ou quebrado (a janela do bug do `tx`, edição 0.64), no instante do aceite para o navegador entrar num estado de descarte **permanente e invisível**.
- **Correção**: `{"ok": false}` passa a significar "reenvie o aceite e tente de novo" — o cliente re-posta o consentimento (o flag local prova que a pessoa aceitou) e repete a navegação, uma vez por carregamento. Sai o `sendBeacon`: beacon não devolve resposta, e sem resposta não há como detectar a dessincronia. Quem já estava dessincronizado **repara sozinho na próxima visita**.
- **Verificação**: estado dessincronizado reproduzido em navegador real, com a sequência observada `telemetry → RECUSADO`, `consent → ok`, `telemetry → ok`.
- **Ressalva de leitura**: os números do Apêndice continuam **subcontando** todo o período anterior a esta correção — soma-se à ressalva já registrada na edição 0.64.
- **IA (A3)**: agente **Claude Code (Anthropic)**; diagnóstico, correção e verificação. Correção (Princípio VII) com decisão humana explícita.

Correção 2026-08-05 · o tutor não conhecia o próprio Radar (spec 077)

- **Defeito relatado pelo editor**: perguntas ao companion sobre um sistema avaliado ("o que acharam do Grok Build?") ou sobre uma apuração do Radar não encontravam nada.
- **Três causas independentes**, todas encontradas no diagnóstico: (1) **escopo** — o índice cobria só `livro/` e o comparativo; as **20 avaliações individuais** e o **Radar inteiro** estavam de fora; (2) **blocos** — markdown não separa linhas de tabela com linha em branco, então a mesa inteira do RADAR virava **um único bloco**, que casava com qualquer pergunta e era truncado em 600 caracteres antes de

chegar ao modelo, cortando justamente a linha procurada; (3) **pontuação** — o score contava cada *ocorrência* de termo sem normalizar por tamanho, então bloco longo vencia bloco curto e exato.

- **Correções**: escopo ampliado (corpus de **738 → 1.406 blocos**), cada linha de tabela virou bloco próprio, e o score passou a contar **termos distintos** da pergunta com divisor logarítmico de tamanho. Verificado por consulta ao vivo: "Grok Build" agora traz a avaliação; "Microsoft Agent Harness" traz o diário do dia; "opencode × Anthropic" traz os dois diários da apuração.
- **Frescor**: o Radar muda todo dia e seu agente só escreve em `radar/` — o CI passou a regenerar e commitar o `corpus.json`, senão o tutor responderia sempre com um dia de atraso.
- **IA (A3)**: agente **Claude Code (Anthropic)**; diagnóstico, correção e verificação. Correção (Princípio VII) com decisão humana explícita.

Edição 0.69 — 2026-08-05 · o jornal ganha capa e acervo (spec 076)

- **Feature spec-kit oficial `076-radar-capa-acervo`**, com o editor no papel de aprovador de uma proposta de UX. O Radar-jornal tinha prazo de validade medido: com uma página só, a fita de abas viraria parede de datas em ~3 semanas e o HTML passaria de **3 MB em um ano**. Agora são duas camadas: **capa** com as 7 edições mais recentes (tamanho constante para sempre) e **acervo mensal** (`radar-AAAA-MM.html`), com o crescimento virando horizontal — nenhuma página passa de ~250 KB, em qualquer horizonte.
- **A capa deixou de ser só cronologia.** Ganhou o **placar da semana** (edições, achados, quantos A/B, quantos promovidos) e a **mesa de edição** — os itens abertos de impacto A/B do `RADAR.md`, que é o dado mais acionável do Radar e não aparecia no site. O diagnóstico que originou a mudança: ninguém pergunta "o que aconteceu em 12 de agosto?"; pergunta "o que mudou no livro?" e "o que espera decisão?".
- **Detalhes de acabamento que a inspeção visual pegou** (e que valem como padrão do projeto): o filtro por impacto mostra **contagem no chip** e desabilita o vazio — filtro que zera a página sem aviso é beco sem saída; e o resumo da mesa corta no travessão sem **partir link de markdown** (o primeiro corte vazava um `[` na tela).
- O contrato do agente do Radar **não mudou**: ele segue escrevendo um arquivo por dia, sem saber que existe acervo.
- **IA (A3)**: agente **Claude Code (Anthropic)** como UX/UI e implementador; decisões do editor (7 dias na capa, só A/B na mesa) e aprovação humana da proposta antes da implementação.

Correção 2026-08-03 · a capa noticiava o dia anterior (spec 075)

- **Defeito relatado pelo editor**: com o Radar do dia publicado e o jornal correto, o card de novidades da capa (e da entrada) ainda mostrava 02/08.
- **Causa raiz**: `noticiaDoRadar()` (em `publicar/build.mjs`) devolvia a **primeira linha válida** da tabela de `radar/RADAR.md`, presumindo que o arquivo estivesse sempre em ordem cronológica reversa. Na varredura de 03/08 as linhas novas entraram abaixo de uma linha de 02/08 (a do Traycer, spec 074) — a ordem física deixou de refletir a cronologia e a capa passou a noticiar um item mais antigo (e **descartado**).
- **Correção**: a notícia passa a ser escolhida por **dado** — data mais recente, desempate por impacto (A > B > C) e depois pela ordem do arquivo. Como higiene, a tabela do RADAR.md foi reordenada por data (nenhuma linha alterada). Verificado com tabela deliberadamente fora de ordem (escolhe 08-03/A ignorando a 08-02 no topo) e no site construído: capa e entrada, PT e EN, em 2026-08-03.
- **Lição registrada**: arquivo mantido por agente agendado não garante ordenação — o motor não deve inferir semântica da posição física. Mesma família do defeito de `tx` (0.64): premissa silenciosa que só falha em produção.
- **IA (A3)**: agente **Claude Code (Anthropic)**; enquadrada como correção (Princípio VII) com decisão humana explícita.

Edição 0.68 — 2026-08-02 · a cadeia de suprimentos vira apêndice — e o teste de inclusão recusa pela primeira vez (spec 074)

- **Novo Apêndice — A cadeia de suprimentos** (PT+EN, nos sumários): o mapa de quem consome quem dentro do corpus, com evidência por elo — QM remendando o Pi com patch de segurança próprio (`package.json:58`), Kimi Code com a TUI do Pi vendorizada, software-agent-sdk orquestrando Codex/gemini-cli via ACP, Grok Build retomando sessões de Claude/Codex/Cursor, e três leituras editoriais (a pergunta "de quem é feito?", a sessão como interface de integração, o enforcement que não viaja pela cadeia).
- **Rodada ext-3**: o **Traycer** (indicação do editor, fork GHDaru/traycer @ `65fc3d7`, MIT) foi avaliado com o instrumento completo e é a **primeira recusa documentada do teste de inclusão** (18/36): ~513 mil linhas abertas de clientes/CLI/protocolo, mas o Host que executa as quatro peças é binário fechado + nuvem obrigatória. A leitura rendeu o mapa dos 18 harnesses que ele orquestra — evidência central do apêndice novo. Corpus permanece em 20.
- **Delta traduzido no mesmo ciclo**: apêndice novo em EN, apêndice do estudo (seção ext-3) espelhado, selos renovados; sumários PT/EN em paridade posicional (29 itens).

- **IA (A3)**: agente **Claude Code (Anthropic)** — avaliador sobre o clone congelado com veredito de inclusão fundamentado; curadoria humana (indicação e aprovação do editor).

Edição 0.67 — 2026-08-02 · rodada ext-2: o corpus vai a vinte — e ganha uma quinta categoria (spec 073)

- **Feature spec-kit oficial `073-ext2-qm-kimi`**: segunda promoção Radar→corpus. **Kimi Code (Moonshot AI, 32/36)** — segundo vendor de modelo verticalizando no harness, com co-design harness↔API (a API do Kimi ganhou capability para servir a *progressive tool disclosure* do harness) e autonomia estruturada (goal mode com budgets, swarm de 128 subagentes) sobre enforcement fraco. **QM (Y Combinator, 31/36)** — não coube na taxonomia e **inaugurou a categoria "agentes organizacionais"**: escopos, contexto filtrado por entitlement da audiência, consentimento de destinatário e auditoria como primitivas; o loop do agente é motor trocável (Pi, OpenCode, Codex, Claude Code) com a sessão portável via "fita".
- Leituras congeladas: fork GHDaru/kimi-code commit `e22479a`; fork GHDaru/qm commit `7f2c916`. Notas no comparativo (leitura da rodada ext-2: polinização cruzada no corpus — a TUI do Kimi Code é fork da do Pi; o QM traz 4 membros do corpus como dependências) e em `notas.json`.
- Livro: caps. 00/01 (vinte sistemas, cinco arquétipos), Apêndice do estudo (seção ext-2), radar (contrato a 20; QM e Kimi Code promovidos). **Delta traduzido no mesmo ciclo** (00/01/apêndice do estudo/comparativo EN, selos renovados).
- **IA (A3)**: agente **Claude Code (Anthropic)** — dois avaliadores em paralelo sobre os clones congelados; curadoria humana (aprovação do editor e revisão das notas).

Edição 0.66 — 2026-08-02 · o jornal atualiza sozinho (spec 072)

- **Feature spec-kit oficial `072-radar-publica-site`** (infraestrutura, 1 linha): `radar/**` entrou nos paths do workflow de publicação — o commit diário do agente do Radar agora reconstrói o site, e o Radar-jornal publica a edição do dia sem esperar o próximo push editorial. Fecha o ciclo da spec 071: apuração agendada → jornal no ar, sem toque humano.
- **Origem**: nota de manutenção da execução agendada de 2026-08-02 — o agente do Radar detectou a lacuna mas não pôde corrigi-la (regra dura: escrita só em `radar/`), registrou e o editor promoveu ("promova"). O contrato funcionando como projetado.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.65 — 2026-08-01 · o Radar vira jornal (spec 071)

- **Feature spec-kit oficial `071-radar-jornal`**: o diário do Radar agora é diagramado como **site de notícias** em `radar.html` — masthead com o contrato editorial, abas por edição, **manchete** (achado de maior impacto do dia), cards com badge A/B/C e **chips de fontes por domínio** (estilo jornalístico: toda afirmação com fonte clicável), e caixas de transparência ("como esta edição foi apurada", "da redação: o que ficou de fora — e por quê", "leituras executivas em risco"). Parser tolerante: diário fora do formato vira matéria corrida — o jornal nunca quebra. Os links "ver o Radar completo" da capa e da entrada (PT/EN) apontam para o jornal.
- **IA (A3)**: agente **Claude Code (Anthropic)** como UX/UI e implementador; validação do editor sobre o site real (hoje + 1 dia).

Edição 0.64 — 2026-08-01 · companion 100% bilíngue (spec 070)

- **Feature spec-kit oficial `070-companion-en`**: fechada a limitação declarada na 067 — **todas** as strings visíveis do widget (paleta, tour, Bastidores, sugestão, BYOK, plano de ensino, tooltips, erros, até o separador decimal dos tokens) passam por `tx(pt, en)`: 120 chamadas, PT byte-idêntico, sem sombreamento de `tx` (a lição do fix de 01/08 virou guarda-corpo do ciclo).
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.63 — 2026-08-01 · revisão de nivelamento: a porta de entrada alargada (spec 069)

- **Feature spec-kit oficial `069-nivelamento`** (origem: parecer editorial simulado para busca de editora): os caps. 00–02 ganharam **chão sem perder teto** — ponte "por que o ChatGPT responde mas não resolve" e seção **"Como ler este livro — três portas de entrada"** (00); a imagem-âncora do "profissional no primeiro dia" + as quatro peças numa tarefa real + preâmbulo leigo da metodologia (01); **um turno completo em câmera lenta** (7 passos) antes do vocabulário técnico (02). Glosas de "janela de contexto" e "tool call" no primeiro uso. Nada removido: as adições são pontes.
- De brinde, a revisão pegou uma desatualização real: os caps. 00/01 ainda diziam **dezesseis** sistemas — corrigido para **dezoito** com Grok Build e Pi nas listas.
- **Delta traduzido no mesmo ciclo** (regra da 067): os 3 capítulos EN espelhados, hashes renovados, selos de sincronia verdes.
- **IA (A3)**: agente **Claude Code (Anthropic)** atuando como editor-revisor; curadoria humana.

Correção 2026-08-01 · o banner de consentimento nunca aparecia

- **Defeito de uma linha, efeito silencioso**: em `publicar/tema/companion.js`, dentro de `montarBanner()`, uma variável local `var tx` sombreava a função `tx(pt, en)` do escopo do módulo. Como `var` é içado para o topo da função, na linha seguinte `tx` já era um elemento DOM, e a chamada `tx("Entendi e aceito", …)` lançava `TypeError` — em **PT e EN**, no carregamento de toda página.
- **Consequência**: `montarBanner()` abortava e, com ele, a `telemetria()` chamada logo depois no mesmo bootstrap. Ou seja, (1) **nenhum leitor novo viu o banner de consentimento** e (2) **nenhum evento de navegação foi registrado** por quem não consentisse pelo painel do chat. **Os números do Apêndice — Uso do livro subcontam** o período anterior a esta correção; leia-os com essa ressalva.
- **Por que passou despercebido**: o cartão de consentimento *dentro* do painel do chat usa a função `tx` correta e seguia funcionando — o caminho testado à mão estava íntegro; o quebrado era o que aparece sozinho, sem clique.
- **Verificação** (Chromium/Playwright sobre o `docs/` construído, antes e depois): antes, `['tx is not a function']` e banner ausente nas duas edições; depois, nenhum erro e banner presente. Build verde: 18 capítulos + aparato em PT e EN.
- **Origem do achado**: o motor foi portado para outro livro vivo, e o mesmo defeito apareceu lá — reuso como forma de teste.
- **IA (A3)**: agente **Claude Code (Anthropic)**; diagnóstico, correção e verificação. Enquadrada como correção trivial (exceção do Princípio VII), com decisão humana explícita.

Edição 0.62 — 2026-07-31 · o livro fala inglês (spec 067)

- **Feature spec-kit oficial `067-livro-en`**: o livro vira **multiidioma** — rodada inglês, em `/en/` espelhado com slugs ingleses. **27 páginas traduzidas** (18 capítulos + benchmark + aparato) por 6 agentes em paralelo sob contrato de tradução (glossário fixo, seções canônicas, estrutura 1:1 verificada, citações em inglês verbatim).
- **PT permanece a fonte canônica; a tradução é artefato derivado com selo de sincronia**: cada fonte EN declara `fonte+edição+hash` do original; o build compara com o PT atual e mostra "in sync" ou o aviso âmbar de tradução atrasada — dívida de tradução é sempre visível, e o portão de qualidade falha se o selo mentir. Regra permanente: toda spec que edite `livro/` inclui o passo "traduzir o delta".
- **UX**: seletor PT·EN (pill textual, sem bandeiras) em todas as páginas, levando à MESMA página no outro idioma; preferência gravada; capa PT com navegador em inglês ganha convite discreto (nunca redirect); `hreflang` correto. Ficam em PT com aviso: Histórico, Radar e o conteúdo do card de news (registros operacionais).
- **Paridade**: PDFs e Markdown completos EN (`harness-engineering.pdf/.md`) no mesmo CI; grafo interativo com rótulos/URLs ingleses; superfície principal do companion em EN (demais strings do widget: limitação conhecida). RAG segue só no PT canônico.
- **IA (A3)**: agente **Claude Code (Anthropic)** (motor i18n + 6 tradutores-agentes); decisões de UX e curadoria humanas.
- 🏷 **Release congelada** (spec 068): GitHub v0.62.0 · **DOI desta versão: 10.5281/zenodo.21724433** (o DOI-conceito 10.5281/zenodo.21632412 segue resolvendo para a versão mais recente). Mecanismo permanente: release = commitar `releases/vX.Y.Z.md` na main (o CI cria tag e Release; o Zenodo cunha o DOI).

Edição 0.61 — 2026-07-31 · leitura integral verificada: as citações agora são do texto (spec 066)

- **Feature spec-kit oficial `066-papers-integrais`**: o editor liberou o arXiv na política de rede do Environment (acesso completo — registrado no diário do Radar como **a única exceção até o momento**, pela dinâmica do livro vivo) e os dois preprints da edição 0.60 foram **relidos na íntegra** pelos mesmos agentes, com mandato de deltas verbatim.
- **Cap. 11 corrigido**: duas frases que circulavam como citação do paper da AI2 eram **paráfrases de agregadores** — substituídas pelo verbatim real (§4.3 e §5.2), com os números das Tabelas 1–3 (evolução de harness piora o GPT-5.4 sem testes unitários; held-out +0,6). Lição de método no diário: paráfrase de agregador vira "citação" em um dia de circulação.
- **Cap. 04 corrigido e ampliado**: ganho do GLM-4.7-Flash era +5,5/+6,8 (não "+3,1"); nuance de baseline explicitada; e o achado pró-harness da Tabela 1 — **trocar só o sumarizador move +6,5 pontos** ("compaction is a performance-critical decision process") — devolve à tese do capítulo o que a "terceira via" parecia tirar.
- **Bibliografia**: os dois itens perdem a ressalva "texto integral pendente"; autores/afiliações completados (Tsinghua/Z.AI; AI2/UW/indep.).
- **IA (A3)**: agente **Claude Code (Anthropic)**; decisão de rede e curadoria humanas.

Edição 0.60 — 2026-07-31 · os papers do Radar entram no livro + conferência A2A (spec 065)

- **Feature spec-kit oficial `065-papers-a2a`**: promoção dos dois itens restantes da varredura de 2026-07-31. Três agentes de leitura; **arXiv bloqueado no ambiente** (registrado no diário do Radar) ⇒ dois papers avaliados pelo abstract com marcação explícita; o survey **lido na íntegra** (62 pp.).
- **Cap. 04** ganha o adendo "a terceira via": CompactionRL — compactação **aprendida no treino** (RL com sumarização no loop, recompensa de tarefa, +7,0 Pass@1 SWE-bench Verified) e a limitação reveladora: ganhos não transferem sem compactação ⇒ acoplamento modelo↔harness — o argumento mais forte até agora para a compactação nativa de provedor.
- **Cap. 11** ganha o adendo "três regras": Rethinking the Evaluation of Harness Evolution (AI2/UW) — evolução automática de harness não supera test-time scaling sob orçamento equiparado; regras (orçamento equiparado, held-out, benchmark sensível a design) adotadas como dever de casa do próprio benchmark do livro (validade convergente).
- **Bibliografia**: 3 itens novos — incluindo o survey **Agent Systems with Harness Engineering** (RUC, maio/2026, OpenReview — a busca o datara de julho; corrigido no diário), com nota de rigor (sem limitações declaradas, sem metodologia de survey, n=3 sistemas) e o mapeamento taxonomia-a-taxonomia (converge no scaffold; diverge em permissões/extensibilidade/interfaces — fortes aqui, futuras lá; treinamento agêntico — forte lá, ausente aqui).
- **Cap. 17**: conferência do A2A concluída — v1.0 já estava coberto; adendo enriquecido (3 camadas, v1.0.1 com **mecanismo formal de extensões**, fonte primária) e a simetria editorial: MCP e A2A chegaram no mesmo trimestre a "extensões formais em vez de features no núcleo".
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.59 — 2026-07-31 · o corpus cresce: Grok Build e Pi (rodada ext-1, spec 064)

- **Feature spec-kit oficial `064-corpus-ext`**: primeira **promoção Radar→corpus** — o Radar achou (varredura de 2026-07-31), o editor aprovou e forkou, dois agentes de leitura varreram os clones congelados e o instrumento padrão (HARNESS_EVAL) foi aplicado. Corpus: 16 → **18**; rodada **ext-1**, sem tocar as fotos das rodadas 1/2.
- **Grok Build (xAI): 35/36** — plataforma máxima; ⭐ em permissões (autorização de shell por **AST** tree-sitter + sandbox kernel-enforced fail-closed), subagentes (**worktrees CoW/BTRFS confirmadas no código**) e extensibilidade (compat poliglota: lê artefatos de Claude/Cursor e porta tools do codex/opencode). Distintivo: workflows Rhai com **replay determinístico**. Gap: zero evals comportamentais.
- **Pi (Earendil): 26/36** — o contraponto minimalista que faltava (caso deliberadamente atípico, lógica de replicação de Yin): 3 em tudo que aceita (compactação ⭐ — a mais completa do corpus; extensibilidade ⭐ com 28 eventos), 0–1 no que recusa por manifesto (MCP/permissões/plan/subagentes) — cada recusa provada por extensão de exemplo testada. A alegação "system prompt <1k tokens" foi **medida**: ~460 tokens na base, mas os AGENTS.md concatenam sem orçamento (~6× o slogan no próprio repo do Pi).
- Livro: caixa "**o contraponto: o harness mínimo**" no cap. 03 + entrada no Apêndice A; adendo "worktrees como infraestrutura" no cap. 10 + entradas no Apêndice A; seção ext-1 no apêndice do estudo; comparativo e `notas.json` com as colunas novas; grafo com os nós grok-build e pi.
- **IA (A3)**: agente **Claude Code (Anthropic)** (leitura de código por subagentes; notas julgadas contra a régua das rodadas 1/2); curadoria e aprovação humanas.

Edição 0.58 — 2026-07-31 · harness-um: o livro inteiro, executável (spec 063)

- **Feature spec-kit oficial `063-harness-um`**: nasce a **implementação de referência** do livro — `harness-um/`, um pacote Python com as features dos capítulos 02–13 num sistema coeso: loop com orçamento (02), contexto em camadas (03), compactação (04), ferramentas com esquema pela assinatura (05), cliente MCP stateless pós-2026-07-28 (06), política permitir/perguntar/negar (07), MEMORIA.md + sessões JSONL (08), plano-artefato (09), subagente só-leitura com contexto limpo (10), verificação pós-mutação (11), ganchos vetáveis + habilidades SKILL.md (12) e REPL (13). **Linguagem ubíqua em português** (o código fala a língua do livro; a borda `provedores.py` é a camada anticorrupção). 19 testes offline via `ProvedorEco`, rodando no CI a cada push.
- **Nome**: "harness-um" (progressão do harness-zero) — decisão do editor após o alerta de colisão: "OpenHarness" já é um sistema do corpus (HKUDS). O apêndice registra a escolha.
- Novo apêndice com a **figura oficial** (núcleo "1" âmbar + anel de 12 segmentos = capítulos 02–13), a tabela da linguagem ubíqua e o "como baixar e rodar".
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria e decisões de nome/hospedagem humanas.

Edição 0.57 — 2026-07-31 · o jornal chega à capa: novidades no splash (spec 062)

- **Feature spec-kit oficial `062-news-capa`**: correção de alvo da 061 — o pedido era a **capa** (`index.html`). O splash agora exibe, entre os CTAs e os créditos: (1) **destaque** — card âmbar `splash-news` com a última notícia do Radar (data, badge de impacto, link

"ver o Radar completo"); (2) **menos destaque** — linha `splash-vedicao` "📖 Nesta edição". Mesmas fontes e mesma postura da 061 (parse falho ⇒ bloco omitido); o bloco da entrada permanece como aprovado. Portão novo: fonte parseia ⇒ a capa noticia.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.56 — 2026-07-31 · news na entrada: a última do Radar + a edição corrente (spec 061)

- **Feature spec-kit oficial `061-news-entrada`**: a entrada do livro ganhou uma faixa de **jornal vivo**, derivada no build sem curadoria extra: (1) **destaque** — card âmbar com a notícia mais recente e relevante do Radar (data, impacto, item com link e "ver o Radar completo"); (2) **menos destaque** — a linha "📖 Nesta edição (vX.Y.0 · data): título — Histórico", parseada da última entrada deste arquivo. **Auto-atualização estrutural**: o agente diário escreve no RADAR ⇒ a capa muda no build seguinte; edição nova aqui ⇒ idem. Parse falhou ⇒ bloco omitido (a entrada nunca quebra).
- Verificação: e2e 4/4 (conteúdo real do MCP 2026-07-28, impacto A, versão e link do Histórico) + screenshot.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.55 — 2026-07-31 · MCP 2026-07-28: o primeiro gatilho extraordinário exercido (spec 060)

- **Feature spec-kit oficial `060-mcp-2026-07-28`**, promovida do **Radar** (diário 2026-07-31, impacto A) — o fluxo do ADR 0007/0008 funcionando de ponta a ponta: aviso → pesquisa com fontes oficiais → registro no radar → spec → revisão.
- **Cap. 06**: nova seção "§6 A guinada stateless — a spec 2026-07-28" (fim do handshake `initialize` e do `Mcp-Session-Id`; MRTR no lugar de sampling/elicitation; extensões formais; cache `ttlMs` como contrato; primeira política de depreciação — 12 meses — cobrindo Sampling/Roots/Logging/HTTP+SSE/DCR→CIMD); **Leitura executiva reescrita** (o que a coorte roda × o que se escreve hoje); "o que roubar" corrigido (fallback SSE agora é depreciado); fonte oficial adicionada; revisão 2026-07-31.
- **Cap. 17**: adendo — a guinada stateless + política de depreciação como sinal de protocolo em fase de infraestrutura. **Etapa 07 do harness-zero**: nota de época na docstring (o handshake ensinado é o protocolo 2025-06). **Cap. 04**: `ttlMs` como o protocolo absorvendo cache. **Glossário/motor**: MRTR, CIMD e DCR. **Bibliografia**: release verificada por fetch direto.
- **IA (A3)**: agente **Claude Code (Anthropic)**; gatilho reportado pelo editor humano; fontes verificadas nesta sessão.

Edição 0.54 — 2026-07-31 · favicon (spec 059)

- **Feature spec-kit `059-favicon`**: o site ganhou favicon na identidade do livro — **núcleo âmbar (o modelo) envolto pelo anel segmentado (o harness)**, a mesma metáfora da capa e do diagrama do cap. 00. `favicon.svg` (nítido em qualquer escala) + PNG 32px + apple-touch-icon 180px, nos dois templates (páginas e splash). Conferido visualmente em 16/32/180px nos dois fundos.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.53 — 2026-07-30 · contador de visitas no rodapé (spec 058)

- **Feature spec-kit oficial `058-contador-visitas`**: o clássico contador de visitas, do jeito honesto — o rodapé de todas as páginas ganha o chip `📈 N visitas registradas`, alimentado pelo **agregado público da telemetria consentida** (`/telemetry/publico`, spec 055), com cache por sessão de leitura (1 requisição/10 min) e **link para o Apêndice — Uso do livro** (o contador como porta de entrada da página de transparência). Sem número na capa e sem "você é a visita #N" — visitantes sem consentimento não contam, então não existe ordinal verdadeiro a atribuir. Backend fora do ar ⇒ o chip simplesmente não aparece.
- Verificação: e2e 6/6 (chip com total e link; cache na 2ª página sem novo fetch; ausência silenciosa sem backend).
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.52 — 2026-07-30 · Knowledge Graph do livro: apêndice interativo, sempre em sincronia (spec 057)

- **Feature spec-kit oficial `057-knowledge-graph`** (ciclo specify→plan→tasks→implement): novo aparato Apêndice — Grafo do livro, com o mapa de conexões do livro **interativo** (força dirigida em canvas, JS puro, zero dependências): 4 tipos de nó (18 capítulos · 16 sistemas do corpus · 6 conceitos · 13 etapas do harness-zero) e arestas com peso = menções reais no texto. Interações: arrasto, zoom, hover, clique (isola a vizinhança + painel com link para a página), filtros por tipo.
- **O sincronismo é estrutural, não processo**: a extração (`publicar/grafo.mjs`) é **determinística, sem LLM**, e roda dentro do `npm run build` — toda mudança publicada do livro regenera o grafo (52 nós / 324 arestas nesta edição). O portão de qualidade agora falha o build se o grafo regredir (18 capítulos, ≥40 nós, ≥100 arestas). Cada aresta é evidência textual verificável (Princípio I aplicado a visualização).
- Verificação: e2e Playwright 7/7 (dados, filtros, clique/painel/link) + screenshots nos 2 temas; build/portão/corpus verdes.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.51 — 2026-07-29 · radar diário: o livro vigia o próprio ecossistema (spec 056)

- **Feature spec-kit oficial `056-radar-diario`** (ADR 0008): uma **sessão-agente agendada (1×/dia)** busca novidades do ecossistema (releases do corpus, protocolos, papers, ferramentas candidatas), avalia impacto por capítulo/Leitura executiva e mantém o **roadmap de auto-atualização** em `radar/RADAR.md`, com o bruto diário auditável em `radar/diario/`. O contrato do agente é versionado em `radar/AGENTE.md` — **escrita somente em `radar/`**; promover item a mudança no livro continua exigindo spec-kit com curadoria humana (a fronteira de autonomia dos caps. 07/16 aplicada ao próprio projeto). O radar é a fila de entrada do gatilho extraordinário do ADR 0007.
- **IA (A3)**: agente **Claude Code (Anthropic)**; decisão registrada em ADR com alternativas.

Edição 0.50 — 2026-07-29 · Apêndice — Uso do livro (vivo) (spec 055)

- **Feature spec-kit oficial `055-apendice-uso-vivo`** (ciclo specify → plan → tasks → implement): o livro passa a **expor a própria telemetria** — novo aparato Apêndice — Uso do livro, com uma **ilha viva** (`data-viz="uso-livro"`, JS puro) que consome o novo `GET /telemetry/publico`: projeção **estritamente agregada** (total, páginas distintas, contagens por página — sem sessões, sem timestamps, por isso pública). A página explica o que é medido e o que não é (consentimento da spec 054, sessões anônimas, direito ao esquecimento) e conecta o painel à cadência do livro vivo (ADR 0007): atenção dos leitores orienta a prioridade de revisão. No PDF a ilha é omitida (regra existente), com aviso no texto.
- Verificação: suíte do backend 14/14 (teste do agregado público sem campos sensíveis); e2e com backend semeado (KPIs, barras, títulos legíveis, nota de privacidade) e fallback honesto sem backend; build/portão/corpus verdes.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.49 — 2026-07-29 · experiência educacional: consentimento, onboarding, telemetria e plano de ensino (spec 054)

- **Feature spec-kit oficial `054-experiencia-educacional`** (ciclo specify → plan → tasks → implement):
  - **Consentimento com aceite gravado**: banner em todas as páginas (e cartão no chat, que fica bloqueado até o aceite) avisa que as conversas alimentam o **aprimoramento vivo do livro** e que **dados pessoais não devem ser compartilhados**; o aceite é versionado e gravado no navegador **e** no backend (tabela `consents`, sessão anônima, `ON DELETE CASCADE` — LGPD preservada).
  - **Onboarding**: tour de 5 passos com spotlight (navegação, cabeçalho/downloads, companion, Bastidores, `/plano`), oferecido após o aceite, 1× por navegador, revisitável com `/tour`; passos sem alvo na página são pulados.
  - **Telemetria de navegação**: só após o aceite (verificado também no servidor), cada página envia `{sessão anônima, slug}` via sendBeacon → tabela `nav_events`; resumo por página em `GET /telemetry` (ADMIN_TOKEN) — insumo de quais capítulos merecem a próxima revisão. Sem IP/UA persistidos.
  - **Objetivo + plano de ensino**: `/plano <objetivo>` grava o objetivo do leitor (tabela `goals`) e pede ao tutor um plano pelos capítulos e etapas do harness-zero; com objetivo gravado, **toda conversa** ganha a camada "Objetivo declarado do leitor" no system prompt (o cap. 03 em ação) e os Bastidores o exibem.
- Verificação: suíte do backend 13/13; e2e Playwright com **14 checagens verdes** (aceite bloqueia/libera, tour navega e não repete, beacon grava só pós-consent, objetivo chega ao prompt e aos Bastidores). Três bugs reais pegos pelo e2e e corrigidos (bootstrap duplo, `[hidden]` × flex, banner sobrepondo o cartão de aceite).
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.48 — 2026-07-29 · chat repaginado: dock, paleta de comandos, tooltips e Bastidores (spec 053)

- **Feature spec-kit oficial `053-chat-ux`** (ciclo completo specify → plan → tasks → implement; gate humano com 3 padrões de tela aprovados): o companion ganhou uma repaginação de usabilidade em quatro frentes:
  - **Layout**: painel flutuante ampliado (480px×78vh) + **modo ancorado** (sidebar direita que empurra o conteúdo — leitura e conversa lado a lado) + **maximizar** (640px); estado persiste entre páginas; mobile vira tela cheia.
  - **Entrada**: textarea de 3 linhas com auto-crescimento, linha de dicas e botão **Enviar** rotulado.
  - **Explicabilidade**: chips de capacidade com **tooltip** (descrição + "libera no cap. X", dados do `/capabilities`); **paleta de `/`** com os cinco comandos descritos, filtro por prefixo e navegação por teclado.
  - **Bastidores** (o livro se demonstrando): barra de status com `~tokens · chamadas · trechos` e painel com Janela de contexto (barra de ocupação), **o que foi injetado no turno** (trechos RAG com fonte), Memória da sessão, e aba **Documentos** (downloads do capítulo + fontes citadas). Backend expõe o bloco `debug` (aditivo) no `/chat` e no stream — tokens sempre estimados (~chars/4), honestamente marcados.

- Verificação: suíte do backend 12/12 (teste novo do `debug`); e2e Playwright com 18 checagens verdes (estados persistem, paleta, tooltip, bastidores com dados reais, regressão de stream/sugestão/BYOK).
- **IA (A3)**: agente **Claude Code (Anthropic)**; padrões de tela aprovados pelo editor.

Edição 0.47 — 2026-07-29 · cadência do livro vivo declarada (spec 052)

- **Feature spec-kit oficial `052-cadencia-livro-vivo`** (ADR 0007): o livro agora tem **política explícita de revisão** — janela **trimestral** (próxima: **2026-10**; re-sync dos 16 forks, diff por dimensão, Apêndices A + placar) e **gatilho extraordinário**: qualquer evento que invalide uma "Leitura executiva" dispara revisão pontual do capítulo, sem esperar a janela. A Leitura executiva (C08) vira o contrato observável de frescor. Guia Editorial ganhou a seção operacional; `publicar/README` atualizado ao estado real do motor; branches mergeadas podadas.
- **IA (A3)**: agente **Claude Code (Anthropic)**; política decidida em ADR com alternativas.

Edição 0.46 — 2026-07-29 · auditoria editorial rodada 2: 27 correções (spec 051)

- **Feature spec-kit oficial `051-auditoria-rodada2`**: 4 auditores (subagentes) leram o livro inteiro em paralelo; 27 achados confirmados e corrigidos. O mais grave: o **cap. 02 estava truncado no meio do Apêndice A desde a reescrita v3** (entrada do IronClaw cortada; Aider, OpenHands, ohmo, n8n e frameworks ausentes) — reconstruído a partir da evidência do benchmark. Demais: cap. 01 §5 realinhado ao corpus real (quatro arquétipos), caps. 15–17 na estrutura do cap. 00, exercícios dos caps. 05/06/07/09/12 realinhados ao harness-zero real, `StorePort` nos caps. 08/13, ACP-IBM desambiguado (cap. 10), contagens do cap. 17 e do glossário corrigidas, e uma dúzia de consertos de português/consistência. Detalhe completo na spec.
- **IA (A3)**: agente **Claude Code (Anthropic)**; achados verificados um a um contra o fonte antes de corrigir.

Edição 0.45 — 2026-07-29 · bibliografia 100% verificada (spec 050)

- **Feature spec-kit oficial `050-bibliografia-verificacao`** (Princípio I — evidência acima de retórica): os **16 itens** ⏳ da Bibliografia foram verificados por **busca web independente nesta sessão** e promovidos a ✓ com dados completos (autores, veículo, páginas, ISBN/DOI). Duas correções encontradas e registradas: o arXiv 2509.18661 é o ***Agentic* AutoSurvey** (Liu et al.), não o AutoSurvey original; e o ISBN 9780226595146 do *Developmental Editing* (Norton) é da **1ª ed. 2009** (a 2ª ed. 2023 tem ISBN 9780226793634). A URL da entrada de Peer Review (Baldwin, CMU ETHOS) foi corrigida. **A fila de pendências da Bibliografia está zerada.**
- **IA (A3)**: agente **Claude Code (Anthropic)**; verificação por busca cruzada com fontes independentes.

Edição 0.44 — 2026-07-29 · rate-limit persistente (spec 049)

- **Feature spec-kit oficial `049-rate-limit-persistente`**: o limite de mensagens **por sessão** agora deriva do **store** (`count_since` sobre as mensagens persistidas — porta que existia desde a spec 016 nos dois adapters): **sobrevive a deploys do Railway e vale entre instâncias**, sem tabela nova. O deque em memória virou guarda secundária **por IP** (`RATE_LIMIT_MSGS × RATE_LIMIT_IP_FACTOR`, default 3×) contra abuso multi-sessão, e segue limitando sugestões. BYOK continua isento. Trade-off registrado: `delete_session` (LGPD) zera a contagem — privacidade > contabilidade; a guarda por IP cobre o atalho.
- Verificação: teste novo simula restart (deque limpo) e o 429 por sessão continua vindo do store; suíte 11/11.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.43 — 2026-07-29 · BYOK no widget (spec 048)

- **Feature spec-kit oficial `048-byok-widget`**: o leitor pode usar a **própria chave de API** no companion — comando `/chave` abre um campo `password` discreto (mesmo padrão sob-demanda da sugestão); a chave fica **só no localStorage do navegador**, mascarada (`…últimos 4`), vai como `byok_key` no `/chat` e `/chat/stream` (o backend já a tratava como efêmera e isenta do rate-limit — specs 016/017), e some com `/chave limpar` ou um clique no selo 🔑 do cabeçalho. A mensagem de limite (429) agora ensina o comando.
- Verificação e2e (uvicorn echo + Chromium): payload com/sem `byok_key` conferido na rede; a chave nunca aparece em texto claro na conversa.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.42 — 2026-07-29 · companion com streaming SSE (spec 047)

- **Feature spec-kit oficial `047-companion-sse`**: as respostas do tutor agora chegam **em streaming** — novo `POST /chat/stream` (`text/event-stream`, eventos `{delta}`/`{trace}`/`{done}`/`{erro}`), `stream()` nos dois adapters de LLM (SSE OpenAI-compatible com agregação de tool_calls por índice; Echo em pedaços, testável sem rede) e `run_turn_stream()` no loop (mesmo freio `MAX_TURNS`, mesmo trace). O widget consome via `fetch`+`ReadableStream`, renderiza incrementalmente (markdown aplicado ao final) e **cai no `/chat` clássico** em falha de transporte — falha do modelo no meio do stream não refaz a chamada (evita duplicar o turno persistido).
- Verificação: suíte do backend 10/10 (novo teste do stream com Echo: deltas + done ≡ histórico persistido); ponta a ponta real com uvicorn local + widget no Chromium (render incremental e markdown final conferidos).
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.41 — 2026-07-29 · a obra ligada ao Awesome Harness Engineering (spec 046)

- **Feature spec-kit oficial `046-awesome-list-obra`** (criada pelo script `.specify/create-new-feature.sh`): a coleção viva **Awesome Harness Engineering** (curada pelo autor, organizada por problema — a mesma taxonomia do livro) agora é referenciada em toda a obra:
  - **"Consulte também"** ao fim das Fontes da indústria dos caps. 02–13 e 17, apontando para a **seção específica** da lista (Agent Loop, Context Delivery, Tool Design, Skills & MCP, Permissions, Memory, Planning, Orchestration, Verification, DX, Human-in-the-Loop…);
  - caps. 14/15/16 (sem seção de fontes): nota antes da Verificação (Foundations, Production Infrastructure, Skills & MCP);
  - cap. 00 (Os harnesses do estudo), cap. 01 §5 e Apêndice do estudo (→ Reference Implementations);
  - **Bibliografia**: nova seção "Coleções vivas" com a entrada da lista.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.40 — 2026-07-29 · download do livro: PDF e Markdown, completo e por capítulo (spec 045)

- **Feature spec-kit oficial `045-downloads`**:
  - **Livro completo**: a entrada ganhou os botões ⬇ **PDF** (`pdf/engenharia-de-harness.pdf`, capa + rodapé paginado) e ⬇ **Markdown** (`md/engenharia-de-harness.md`, concatenação na ordem do sumário com cabeçalho de versão/DOI — útil inclusive para alimentar LLMs).
  - **Por capítulo**: o cabeçalho de cada capítulo (C01) ganhou os chips ⬇ **md** (fonte exata) e ⬇ **pdf** (avulso com título, datação e rodapé paginado) — 18 PDFs gerados no CI.
  - **Correção**: o PDF completo tinha perdido os títulos de capítulo após a spec 043 (o `h1` saiu do `<article>`); o gerador agora injeta o título do sumário + linha de datação do cabeçalho. O painel Leitura executiva (C08) também ganhou estilo de impressão.
  - **CI**: o workflow instala Chromium (Playwright) e gera os PDFs após o build; o portão por capítulo confere links e artefatos de download.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.39 — 2026-07-29 · correções do editor: contagem de arquétipos + sugestão sob demanda + Gmail

- **Feature spec-kit oficial `044-correcoes-cap00-sugestoes`**:
  - **Cap. 00**: a contagem agora bate com a lista — "**quatro** arquétipos" (código, agentes pessoais self-hosted, embutidos, frameworks). A taxonomia própria do cap. 01 §5 (três, com 3 itens) permanece.
  - **Companion — sugestão sob demanda**: o formulário "enviar ao autor" **não aparece mais por default** (corrigido inclusive um bug de CSS que o deixava sempre visível) e o botão 💡 permanente saiu do cabeçalho. Ele abre quando o leitor pede no chat: comando `/sugerir` ou intenção explícita ("quero enviar uma sugestão ao autor"); a solicitação é resolvida no widget, sem passar pelo tutor. As boas-vindas mencionam o comando.
  - **Email via Gmail**: `chat-companion/backend/EMAIL.md` documenta a configuração (senha de app do Google + variáveis `SMTP_*` no Railway; remetente = conta do autor, destinatário = `SUGGESTION_EMAIL_TO`). Nenhuma credencial no repositório.
- **IA (A3)**: agente **Claude Code (Anthropic)**; revisão editorial humana.

Edição 0.38 — 2026-07-28 · design system dos capítulos: C01 + C08 + N02 (spec 043)

- **Feature spec-kit oficial `043-template-capitulos`** (ADR 0005 e ADR 0006): o catálogo de componentes (`publicar/DESIGN-SISTEMA.md`) ganhou os três componentes que faltavam, todos aprovados em **gate humano** (página-espécime + 3 modelos por componente):
  - **C01 CabeçalhoDeCapítulo — variante B "faixa editorial"**: kicker da parte, título, teaser, número em marca d'água, datação absorvida (C02) e **tempo de leitura estimado**; o `h1` e o blockquote de datação do Markdown saem do corpo (sem duplicação). Só páginas de capítulo numeradas; o aparato mantém o selo clássico.
  - **C08 LeituraExecutiva — V1 "painel âmbar"**: a seção `### Leitura executiva` (16 capítulos) vira painel destacado com rótulo em versalete; âncora preservada.
  - **N02 PaginaçãoEmCartões — V2 "cartões com badge"**: anterior/próximo na linguagem dos cartões da entrada (badge numerado = "clique para ir a um capítulo").
- **Portão novo por capítulo** (ADR 0005): `publicar/verifica-capitulos.mjs` confere os 18 capítulos (badge correto, `h1` único, datação absorvida, C08 aplicado) e as 7 páginas de aparato — falha encerra com erro.
- **IA (A3)**: agente **Claude Code (Anthropic)**; direção de arte e aprovações (B/V1/V2) humanas.

Edição 0.37 — 2026-07-28 · harness-zero: etapa 12 — skills (cap. 16) — TRILHA COMPLETA

- **Feature spec-kit oficial `042-harness-zero-etapa12`**: o harness que **aprende — com freio**. `salvar_skill(nome, quando_usar, conteudo)` captura procedimentos como skills, mas a skill **nunca entra em vigor sozinha**: vai para `skills/pendentes/` (**auto-aprovação = prompt injection persistente**, o anti-padrão central do cap. 16); o humano **aprova** (`POST /skills/aprovar`) ou rejeita. Aprovada, entra como **camada nova do MontadorDeContexto** (etapa 03 pagando dividendos) — **só nome + quando usar** no prompt; o conteúdo completo vem sob demanda via `ler_skill` (**progressive disclosure**, cap. 04). Smoke com asserções: pendente fora do contexto; aprovada dentro (índice apenas); conteúdo via tool.
- 🏁 **Com esta etapa, a trilha prática fecha o mapa completo: etapas 00–12** — loop, tools (schemas derivados), contexto em camadas, sessões, compactação, permissões+aprovação, MCP, plan mode, subagentes, evals (replay+juiz), hooks e skills. As doze dimensões do livro, construídas do zero, cada etapa autocontida e verificada.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.36 — 2026-07-28 · harness-zero: etapa 11 — hooks (cap. 12)

- **Feature spec-kit oficial `041-harness-zero-etapa11`**: extensibilidade sem tocar no loop. **Hooks** em duas fronteiras estáveis da execução de ferramentas — `pre_tool` (pode **bloquear** ou **ajustar args**) e `post_tool` (pode **transformar o resultado**) — envolvendo `registro.executar` em todos os caminhos (loop, aprovação, subagente). Dois hooks de exemplo com dor real: **auditoria** (cada chamada vira linha estruturada em `auditoria.jsonl`; janela `GET /auditoria`) e **redator** (mascara padrões de segredo — `nvapi-…`, `password=` — antes de o resultado chegar ao modelo; defesa em profundidade somada à política da etapa 06). Smoke: redação, bloqueio e log verificados.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.35 — 2026-07-28 · harness-zero: etapa 10 — evals do harness (cap. 11)

- **Feature spec-kit oficial `040-harness-zero-etapa10`**: o harness aplicado a si mesmo. Suíte `evals/` com **`ReplayAdapter`** — **respostas gravadas** em `.jsonl` reproduzidas em ordem: o eval testa o **harness** (política, plan mode, escada de compactação, derivação de schema, **pausa de aprovação**: a gravação pede `write_file` e nada é escrito sem o humano), nunca o humor do modelo. **`juiz.py`** — LLM-as-judge atrás do mesmo `LLMPort` (nota+justificativa por critérios; com echo degrada honestamente, com chave real julga). 6/6 verdes.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.34 — 2026-07-28 · harness-zero: etapa 09 — subagentes (cap. 10)

- **Feature spec-kit oficial `039-harness-zero-etapa09`**: a tool **`task(descricao)`** delega a um **subagente com sessão-filha** (`task-…`): system prompt focado **só na descrição** (zero contexto do pai), **mesmo loop** com turnos limitados, ferramentas **restritas a leitura** (filha não muta o mundo; a política da etapa 06 segue por cima) — e **só o resultado final volta** ao pai como tool result. As duas fronteiras (ida: só a descrição; volta: só o resultado) são a lição do capítulo: é o que mantém o contexto do pai limpo. Filhas persistidas e visíveis em `/sessions` (a etapa 04 pagando dividendos). Smoke com asserções nas fronteiras.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.33 — 2026-07-28 · harness-zero: etapa 08 — plan mode (cap. 09)

- **Feature spec-kit oficial `038-harness-zero-etapa08`**: plan mode **imposto, não pedido**. Um **modo por sessão** (`executar`/`planejar`, `POST /modo`); em `planejar`, a **política da etapa 06 nega toda ferramenta mutante** — a mudança é **uma linha** no `decide()` (a lição: quem garante o comportamento é o mecanismo de permissões, não a boa vontade do modelo). Nova tool `propor_plano` grava o artefato **PLAN.md** revisável (`GET /plano`); aprovar o plano = trocar o modo para executar. O turno em modo planejar recebe o aviso injetado. Smoke: negação com motivo em planejar; executar volta a perguntar.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.32 — 2026-07-28 · harness-zero: etapa 07 — MCP client (cap. 06)

- **Feature spec-kit oficial `037-harness-zero-etapa07`**: o harness aprende a **plugar ferramentas dos outros**. A etapa traz um **servidor MCP de exemplo** (~60 linhas, JSON-RPC 2.0 por linha no stdio — para o leitor ver o protocolo por dentro) e o **ClienteMCP** no harness: `initialize` → `tools/list` → `tools/call`, com as tools importadas de **prefixo `mcp_`** num **RegistroComposto** (locais + MCP atrás da mesma interface — o loop não sabe de onde a ferramenta vem). A **política da etapa 06 vale para as tools MCP** (servidor externo é input não-confiável); trace distingue 🔧 local × 🔌 MCP; degradação graciosa se o servidor cair. Smoke: handshake + list + call verificados.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.31 — 2026-07-28 · harness-zero: etapa 06 — permissões (cap. 07)

- **Feature spec-kit oficial `036-harness-zero-etapa06`**: fecha a **ferida aberta desde a etapa 1** (`read_file` lia qualquer arquivo, inclusive `.env`). Nasce a **PermissionPolicy** como **domínio puro** — `decide(tool, args) → permitir | perguntar | negar`, uma função sem I/O — com **paths sensíveis fixos no código** (segurança que o usuário pode desligar não é segurança) e `write_file` exigindo **aprovação humana inline**: o turno **pausa** (pendência com id), o chat mostra [aprovar]/[negar], e o loop **retoma do ponto exato**. Negação vira **texto para o modelo** (ele explica e segue). Evolução justificada do chat congelado (a aprovação exige superfície). Smoke: política pura + pausa/aprovação/retomada verificadas.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.30 — 2026-07-28 · harness-zero: etapa 05 — compactação (cap. 04)

- **Feature spec-kit oficial `035-harness-zero-etapa05`**: a **etapa 05** (`harness-zero/etapas/05-compactacao/`) implementa a **escada de agressividade** do cap. 04: degrau 1 **trunca** resultados de ferramenta antigos, degrau 2 **poda** turnos antigos, degrau 3 **sumariza** o podado via `LLMPort` e injeta o resumo — acionada por **orçamento** de contexto (chars como proxy didático de tokens; `ORCAMENTO_CHARS` para experimentar). Lições materializadas: a escada age na **visão** enviada ao modelo, nunca no **registro** persistido (etapa 4 intacta), e **compactação avisa** (indicador 🗜 no trace — silenciosa é dívida invisível). Janela `GET /contexto_uso`. Smoke verificado (degraus 2 e 3 disparando).
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.29 — 2026-07-28 · harness-zero: etapa 04 — sessões (cap. 08)

- **Feature spec-kit oficial `034-harness-zero-etapa04`**: a **etapa 04** (`harness-zero/etapas/04-sessoes/`) paga a dívida carregada de propósito desde a etapa 0: o histórico sai da variável global. Nasce o **StorePort** (terceira porta) com dois adapters — `MemoriaStore` (o contraste didático) e **`SQLiteStore`** (persistência real: converse, mate o servidor, volte — a conversa fica). Conceito de **sessão** (`session_id` + `/sessions` + `/history` = o *resume* dos harnesses reais); **1ª evolução justificada do chat congelado** (a dimensão exigiu superfície: id no navegador + retomada do histórico). O companion roda a mesma arquitetura em produção (mesmo StorePort, adapter Postgres). Smoke verificado.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.28 — 2026-07-28 · harness-zero: etapa 03 — contexto em camadas (cap. 03)

- **Feature spec-kit oficial `033-harness-zero-etapa03`**: a **etapa 03** (`harness-zero/etapas/03-contexto/`) introduz o **MontadorDeContexto** — o system prompt montado em **camadas nomeadas** (identidade fixa → ambiente derivado → **regras do projeto via `AGENTS.md`**), **remontado a cada turno** (edite o AGENTS.md com o chat aberto e veja o comportamento mudar sem redeploy — o artefato-padrão do cap. 01 §9 em ação). Loop e ToolPort intactos (mudança de uma linha no loop); janela de observação `GET /contexto` mostra as camadas e o prompt final; o EchoAdapter passou a exibir o tamanho do system prompt. Etapa autocontida, roda sem rede.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.27 — 2026-07-28 · harness-zero: etapa 02 — ToolPort (cap. 05)

- **Feature spec-kit oficial `032-harness-zero-etapa02`**: retomada a trilha prática. A **etapa 02** (`harness-zero/etapas/02-tools/`) introduz o **ToolPort** — a segunda porta do harness: ferramentas são **funções Python tipadas** registradas por decorator (`@tools.tool`); o **JSON Schema é derivado** da assinatura + docstring (`inspect`/`typing`), curando o tédio dos schemas à mão da etapa 1 (a mesma solução dos harnesses reais: FastMCP, `function_tool`, `#[tool]`). O loop não mudou — é assim que uma porta paga o aluguel. Endpoint `/tools` como janela de observação; parâmetros com default viram opcionais no schema (verificado). Etapa autocontida; roda com echo (sem rede).
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.26 — 2026-07-28 · companion: corpus atualizado + apagar conversa

- **Feature spec-kit oficial `031-companion-atualizacoes`**: o **`corpus.json`** do companion foi **regenerado** (618 blocos) — o tutor volta a citar o livro **atual** (Fundamentos novo, Glossário, Apêndice do estudo, caps 14–17 revisados). Prática registrada: regenerar o corpus a cada mudança relevante do livro (roadmap R2; automação via CI fica como evolução).
- **Widget**: novo botão **🗑 Apagar a conversa** (CO2 do roadmap) — confirma, chama `DELETE /session/{id}` (LGPD) e reinicia o chat localmente.
- **IA (A3)**: agente **Claude Code (Anthropic)**; curadoria humana.

Edição 0.25 — 2026-07-28 · formato editorial v3 nos caps 00, 14–17 + siglas inline (auditoria)

- **Feature spec-kit oficial `030-formato-editorial`** (O005 da auditoria): os capítulos **pré-v3** foram trazidos ao formato editorial do livro (padrão do cap. 04): **14 — Convergências**, **15 — Harness Embutido**, **16 — Aprendizado e Auto-melhoria** e **17 — Protocolos** ganharam cabeçalho de data, objetivos de aprendizagem (Bloom), "O problema", estado da arte reorganizado com leitura executiva, verificação e Apêndice A (material por-repositório preservado, com link às avaliações). **Conteúdo preservado; nenhuma fonte inventada** (capítulos sem papers não ganharam seção de fundamentos — pendência honesta).
- **Cap. 00 (Introdução)**: cabeçalho de data + seção **"Os harnesses do estudo"** (O004): a **lista completa dos 16 sistemas** por arquétipo, com ponteiro ao Apêndice — O estudo e ao Comparativo (substitui a antiga "primeira rodada").
- **Siglas por extenso inline (O003)**: 1ª ocorrência de cada sigla técnica agora traz o nome por extenso no próprio texto dos caps 00–13 (46 expansões aplicadas; casos que quebravam a leitura foram tratados manualmente). O `<abbr>` continua cobrindo todas as demais ocorrências.
- **IA (A3)**: agente **Claude Code (Anthropic)** — revisão editorial (4 sub-editores em paralelo p/ 14–17) sob as regras "preserve o conteúdo; não invente fontes"; curadoria humana.

Edição 0.24 — 2026-07-28 · PDF do livro (E08)

- **Feature spec-kit oficial `029-pdf-livro`**: novo gerador **`publicar/pdf.mjs`** — produz o **PDF completo do livro** (`docs/engenharia-de-harness.pdf`) a partir do site construído: folha de rosto (capa, autor+co-autoria de IA, versão, data, DOI), todas as partes e capítulos na ordem do sumário, CSS de impressão (A4, quebras por capítulo, rodapé com paginação). Uso: `node build.mjs && node pdf.mjs`. O PDF é artefato gerado (não versionado); pode ser anexado a cada Release/DOI.
- **IA (A3)**: agente **Claude Code (Anthropic)** — implementação; curadoria humana.

Edição 0.23 — 2026-07-28 · Apêndice "O estudo" + fork/commit por harness + citações→Bibliografia

- **Feature spec-kit oficial `028-estudo-citacoes`** (E01+E03+E04 da auditoria): nova página **Apêndice — O estudo** (`livro/apendice-estudo.md`), no aparato: os **16 harnesses avaliados** com **origem**, versão/snapshot, **fork GHDaru + commit lido** (a data de corte materializada — reprodutibilidade do método, cap. 01 §6), data/rodada e link para a **avaliação completa** de cada um; mais o template (`HARNESS_EVAL`/`FRAMEWORK_EVAL`) e a ponte para o Comparativo.
- **Citações (MVP)**: o motor agora converte menções textuais `arXiv NNNN.NNNNN` em **link para a Bibliografia** (que linka as fontes). Decisão e evolução planejada em `adr/0004`.
- **IA (A3)**: agente **Claude Code (Anthropic)** — extração dos metadados reais das avaliações e implementação; curadoria humana.

Edição 0.22 — 2026-07-28 · ilustração esquemática do harness (E02)

- **Feature spec-kit oficial `027-ilustracao-harness`**: o cap. 00 ganhou uma **figura esquemática (SVG flat, estilo bloco)** — o **modelo no centro** e, em volta, os seis blocos do harness (loop, contexto, ferramentas, memória, permissões, verificação) numa

moldura "HARNESS (o andaime)", com o mundo (arquivos/APIs/terminal) à direita. **Theme-aware** (herda as cores do tema via CSS vars), acessível (`<title>`/`alt`/`figcaption`), sem binário (SVG versionável).
- **IA (A3)**: agente **Claude Code (Anthropic)** — desenho e integração; direção do autor ("menos futurista, mais bloco").

Edição 0.21 — 2026-07-28 · companion: sugestões dos leitores (E05)

- **Feature spec-kit oficial `026-companion-sugestoes`**: o leitor agora pode **enviar sugestões ao autor pelo chat** (botão 💡 no widget). O backend persiste em `suggestions` (Postgres/memória) **antes** de qualquer coisa e envia **email** ao autor quando SMTP está configurado (env; instruções no `.env.example` — Gmail com App Password). Sem SMTP, o autor lê via `GET /suggestions?token=` (`ADMIN_TOKEN`). Rate-limit aplicado; nenhuma sugestão se perde por falha de email.
- **IA (A3)**: agente **Claude Code (Anthropic)** — implementação e testes (9/9 verdes); curadoria humana.

Edição 0.20 — 2026-07-28 · Fundamentos reescrito com história e método (rigor)

- **Feature spec-kit oficial `025-fundamentos-rigor`** (a pedido do editor: "fundamentos fraco; falta rigor metodológico/científico"): o **cap. 01** foi reescrito em 9 seções — definição (com *scaffolding* = **andaime** traduzido e introduzido), **o que havia antes** (sistemas especialistas, RPA, chatbots, Copilot-autocomplete e por que não eram agentes), a **linhagem técnica** (CoT → **ReAct** → function calling → AutoGPT/BabyAGI e sua lição → CLIs de código → protocolos MCP/A2A/AGENTS.md) com **linha do tempo**, a definição constitutiva (4 elementos), a **proveniência do corpus** (3 arquétipos + teste de inclusão), e a nova seção **"O método do estudo"**: casos múltiplos (Yin) + Mining Software Repositories (Hassan 2008) + GQM (Basili) + feature analysis DESMET (Kitchenham 1997) + benchmarking científico (Sim et al. 2003) + Design Science (Hevner 2004; Peffers 2007) + **tabela de ameaças à validade** (Cook & Campbell), em que a cláusula de expiração vira mitigação declarada.
- **Bibliografia**: novas seções "História e proveniência" e "Metodologia do estudo" com fontes ✓ **verificadas** por pesquisa dedicada (ReAct arXiv 2210.03629; anúncios primários de Copilot/function calling/MCP/A2A; Hassan; Runeson & Höst; DESMET; Sim; Stol; Peffers) e itens ⏳ a confirmar explicitamente marcados (Princípio I).
- **Decisão registrada**: `adr/0003-fundamentos-rigor.md` (alternativas avaliadas e justificativa).
- **IA (A3)**: agente **Claude Code (Anthropic)** — pesquisa (2 frentes verificadas) e redação; direção editorial humana.

Edição 0.19 — 2026-07-27 · foto + LinkedIn do autor; LinkedIn na capa

- **Feature spec-kit oficial `024-autor-linkedin`** (E06+E07 da auditoria): a página "Sobre o autor" ganhou a **foto** do autor (`assets/autor.png`, flutuando à direita, responsiva) e a **tela-capa** passou a incluir o link do **LinkedIn** nos créditos (repositório é público).
- **IA (A3)**: agente **Claude Code (Anthropic)** — implementação; foto enviada pelo autor.

Edição 0.18 — 2026-07-27 · Glossário + siglas por extenso

- **Feature spec-kit oficial `023-glossario-siglas`**: nova página **Glossário** (`livro/glossario.md`) com as siglas do livro **por extenso**, explicação curta e **em que capítulos aparecem** (agrupadas por tema). Fiel ao texto (siglas varridas; expansões conferidas na fonte — Princípio I).
- **Siglas "abertas" em todo o livro**: o motor envolve automaticamente cada sigla conhecida em `<abbr title="Por Extenso">` — o leitor vê o significado ao passar o mouse — de forma **não-invasiva** (sem mexer no Markdown-fonte) e **HTML-safe** (não toca em código, `<pre>`, links ou títulos).
- **Política no Guia Editorial**: expandir na 1ª ocorrência; o mapa de siglas vive no motor e é espelhado no Glossário.
- **IA (A3)**: agente **Claude Code (Anthropic)** — implementação; curadoria e aprovação humanas.

Edição 0.17 — 2026-07-27 · experiência de entrada do livro (índice repaginado)

- **Feature spec-kit oficial `021-experiencia-entrada`**: o sumário deixou de ser uma lista crua e virou uma **entrada de verdade** — mantendo a **sidebar** com o índice completo (navegação sem rolar), o conteúdo principal ganhou: **hero** (capa + título + `vX.Y.0`/DOI + CTAs), card **"Continue lendo/Retomar"** (via `localStorage`, aparece após ler um capítulo), **trilha** em 4 passos (Fundamentos → Funcionalidades → Benchmark → Mão na massa) e os **capítulos em cartões** com *teaser*; benchmark/aparato/sobre como **pills**.
- **Teasers por capítulo** entraram no `sumario.json` (conteúdo reaproveitável). O motor grava o último capítulo lido e popula o "Retomar".
- **Theme-aware** (claro/escuro via `--vars`), **responsivo** (hero empilha, trilha 2 col., cartões 1 col. no mobile) e acessível. O cartão vira a **base do template dos capítulos** (feature futura).

- **IA (A3)**: agente **Claude Code (Anthropic)** — design e implementação; curadoria e aprovação humanas (mockups revisados antes de publicar).

Edição 0.16 — 2026-07-27 · fix: itálico no markdown do chat-companion

- **Feature spec-kit oficial `022-companion-markdown`**: o widget do companion agora renderiza **itálico** `*x*` (antes vazava como asteriscos). `fmt()` converte `*itálico*` em `<em>` após o negrito, **sem** tocar em `**` nem quebrar identificadores `snake_case`. Escape antes da formatação mantido (segurança).
- **IA (A3)**: agente **Claude Code (Anthropic)** — correção; curadoria humana.

Edição 0.15 — 2026-07-27 · DOI emitido e fixado

- **Feature spec-kit oficial `019-doi-badge-site`**: o **DOI** da obra foi emitido pelo Zenodo — **10.5281/zenodo.21632412** — e fixado: **badge** no README, **link do DOI** na tela-capa (junto ao selo de versão) e seção **"Como citar"** na página do autor.
- Com isso, a obra passa a ser **citável academicamente** com identificador persistente, versionado por edição — a cláusula de expiração agora tem um DOI.
- **IA (A3)**: agente **Claude Code (Anthropic)** — fixação do DOI; curadoria humana.

Edição 0.14 — 2026-07-27 · preparação de DOI e citação (Zenodo/DataCite)

- **Feature spec-kit oficial `018-doi-citacao-zenodo`**: repositório preparado para receber um **DOI** via **Zenodo** (DataCite) — modelo de **concept DOI** (obra viva) + **DOI por versão** (cada edição), espelhando a cláusula de expiração.
- **Licenciamento duplo**: `LICENSE` = **CC BY 4.0** (conteúdo) e `LICENSE-CODE` = **MIT** (código), com nota no README dizendo o que cada uma cobre.
- **Metadados de citação**: `CITATION.cff` (o GitHub passa a mostrar "Cite this repository") e `.zenodo.json` (autor **Gilsiley Henrique Darú** + ORCID `0000-0002-8979-0461`, tipo = livro, licença, keywords, idioma, links para o site). A **co-autoria de IA** é declarada na descrição, **não** como creator (ICMJE/COPE, Guia §6).
- **README**: seções "Como citar" (com espaço para o badge do DOI) e "Licença".
- **Pendente (follow-up)**: o autor liga o Zenodo ao repo e publica um *release* → o DOI é emitido; então o **número/badge** é fixado no README e na capa/colofão do site.
- **IA (A3)**: agente **Claude Code (Anthropic)** — preparação dos metadados; curadoria humana.

Edição 0.13 — 2026-07-27 · chat-companion: widget no site

- **Feature spec-kit oficial `017-widget-chat-companion`**: o **widget** do companion — um chat flutuante (launcher que abre/minimiza) presente em **todas as páginas, inclusive a capa**. JS/CSS puro injetado pelo motor `publicar/` (progressive enhancement; sem JS a página segue inteira).
- **Cabeçalho de capacidades por capítulo**: o painel mostra "o que posso fazer agora (até o cap. N)" com as capacidades **ativas** (verdes) e as **bloqueadas** (🔒), conforme o capítulo da página e o modo (avançado × progressivo). O capítulo é derivado no build a partir do título; o mapa de capacidades é espelhado no build para render instantâneo — o **backend continua impondo** o gating no `/chat`.
- **Conversa e memória**: fala com o backend (016) em `POST /chat`; identidade **anônima por navegador** (`localStorage`), com histórico via `GET /history`. Degradação graciosa se o backend cair (aviso amigável; a página nunca trava).
- **Acessível e responsivo**: `aria-label`, foco ao abrir, teclado (Enter envia, Esc fecha), contraste; painel quase full no mobile; legível sobre a capa escura; theme-aware.
- **Backend no ar**: publicado no Railway (`harnessengineering-production.up.railway.app`) com Postgres (Neon) e NVIDIA NIM; `/health` = `openai`+`postgres`; `/chat` já cita o livro.
- **IA (A3)**: agente **Claude Code (Anthropic)** — implementação; curadoria humana.

Edição 0.12 — 2026-07-27 · chat-companion: backend (harness-zero ao vivo)

- **Feature spec-kit oficial `016-chat-companion-backend`**: nasce o **backend do chat-companion** em `chat-companion/backend/` — um serviço FastAPI que **é o harness-zero rodando em produção** (reusa `LLMPort` e o loop de tool-calling do etapa 01). Atende o futuro widget do site.
- **Portas (hexagonal por necessidade)**: `LLMPort` (echo / OpenAI-compatible → NVIDIA NIM, com **BYOK** por requisição), `StorePort` (`MemoryStore` para dev / `PostgresStore` para **Neon**, com criação de tabelas na subida) e `ToolPort` (tools **seguras/sandbox**: hora, cálculo aritmético seguro, busca no texto do livro).

- **Gating de capacidades por capítulo** (`capabilities.py`): modo **avançado** (tudo) × **progressivo** (só o que o livro ensinou até o capítulo atual) — o *fading* do 4C/ID virando comportamento. `GET /capabilities` é a fonte que o widget exibe ("o que posso fazer agora").
- **Endpoints**: `/health`, `/capabilities`, `/session`, `/chat`, `/history`, `DELETE /session/{id}` (LGPD). **Identidade anônima** por navegador; **rate limit** por sessão/IP (BYOK isenta); **CORS** restrito.
- **Segurança (cap. 07 aplicado a si)**: nenhum segredo no repo; chave só em env; `.env` gitignored; tools sandbox; BYOK nunca persistida. Suíte de smoke (echo + memória) verde, **sem rede e sem banco**.
- **Deploy**: artefatos (`Procfile`, `railway.json`, `runtime.txt`, `requirements.txt`, `.env.example`) e **README com passo-a-passo Neon + Railway**. O deploy do Railway é manual do autor; o Pages não hospeda o backend.
- **Tensão intencional documentada**: o companion (produção) roda à frente das etapas didáticas — `StorePort`/`ToolPort` que as etapas 02/04 formalizarão depois. Registrado no plano, não é violação.
- **IA (A3)**: agente **Claude Code (Anthropic)** — arquitetura, código e testes; curadoria humana.

Edição 0.11 — 2026-07-27 · versão e data de atualização na tela-capa

- **Feature spec-kit oficial `015-versao-data-capa`**: a tela-capa (splash) passa a exibir um selo discreto **`vX.Y.0 · atualizado em <data>`**. A **versão** é derivada automaticamente da **última edição deste histórico** (fonte única — `### Edição X.Y` → `vX.Y.0`), de modo que o placar de edições e a versão exibida nunca divergem. A **data** vem do **último commit** no momento do build (`git log -1`), fiel à última modificação de conteúdo; sem git, cai para a data do build. Fallbacks totais: o selo jamais quebra o build nem o gate de link-check.
- **Coerência com a tese**: carimbar versão + data de atualização logo na entrada materializa a cláusula de expiração (livro vivo) na própria porta do site.
- **IA (A3)**: agente **Claude Code (Anthropic)** — implementação; curadoria humana.

Edição 0.10 — 2026-07-27 · página "Sobre o autor"

- **Feature spec-kit oficial `014-pagina-sobre-autor`**: nova página de *back matter* **"Sobre o autor"** (`livro/autor.md` → `autor.html`), com a biografia acadêmica e profissional de **Gilsiley Henrique Darú** — formação (doutorado UFPR em andamento, mestrados USP e UFPR, especializações), atuação profissional (Neogrid: Head de Dados & IA e trajetória no laboratório de inovação; WEG, Malwee, Datasul), docência (professor universitário na UDESC e outras; coordenação de curso de Engenharia de Produção na FAMEG; pós-graduação em IA & Deep Learning) e produção acadêmica (artigos, anais, orientações), com perfis verificáveis.
- **Navegação**: item entra no `sumario.json` (parte "Sobre"), aparecendo na sidebar e no sumário, com paginação padrão; o nome do autor nos **créditos da tela-capa** vira link para a página.
- **Fontes**: Currículo Lattes (`6253911800847523`), ORCID (`0000-0002-8979-0461`), perfil profissional público (LinkedIn) e busca web verificável (Journal of Lean Systems, art. 1930). Fatos rastreáveis, sem dados inventados (Princípio I); empresas/instituições citadas como trajetória, sem endosso (Princípio VI).
- **IA (A3)**: agente **Claude Code (Anthropic)** — pesquisa das fontes, redação e implementação; curadoria e responsabilidade humanas.

Edição 0.9 — 2026-07-27 · tela-capa full-screen (splash)

- **Feature spec-kit oficial `013-splash-capa-cheia`**: `index.html` virou uma **tela-capa full-screen** (capa grande + título + subtítulo + créditos + CTA "Entrar no livro"), sem sidebar; o índice migrou para **`sumario.html`** (com a navegação). A marca das páginas internas aponta para o sumário e há link discreto para a capa; paginação Sumário↔capítulos. Responsiva, `alt` descritivo, gate de link-check verde.
- **IA (A3)**: agente **Claude Code (Anthropic)** — implementação; imagem por **GPT (OpenAI)**; curadoria humana.

Edição 0.8 — 2026-07-27 · capa e landing (hero) no site

- **Feature spec-kit oficial `012-landing-capa`**: a home (`index.html`) ganhou uma **hero de capa** com a imagem gerada (`capa.png`, 1024×1536), título, subtítulo, CTAs ("Começar a ler", Benchmark, Guia) e **créditos como texto** (Gilsiley Henrique Darú — edição/direção/orquestração; Claude/Anthropic — pesquisa/texto; GPT/OpenAI — imagem); o sumário permanece abaixo. Responsiva (empilha e vem antes da navegação no mobile), theme-aware, com `alt` descritivo.
- **Preview social**: meta tags Open Graph + `capa-social.png` (1200×630, gerada via Chromium) para previews de link.
- **Motor**: `build.mjs` copia os assets de capa e injeta as meta OG; sem quebra do gate de link-check.

- **IA (A3)**: agente **Claude Code (Anthropic)** — pesquisa/texto e implementação; imagem de capa por **GPT (OpenAI)**; curadoria e responsabilidade humanas.

Edição 0.7 — 2026-07-26 · emenda de constituição v1.2.0 (achados do Guia §6)

- **Governança (emenda direta, exceção do Princípio VII, registrada aqui):** constituição **v1.1.0 → v1.2.0** incorporando dois achados do estudo de metodologias (parecer `estudos/2026-07-26-achados-metodologia-escrita.md`):
  - **A2 — revisão developmental** vira portão de qualidade: antes do copyedit, um passo de re-ver estrutura e sentido ("escrever é reescrever"; Sommers/Flower-Hayes). Refletido no Guia §6.E (fluxo) e na seção de portões de qualidade.
  - **A3 — registro do modelo de IA** na datação (Princípio IV): toda edição registra o agente/modelo de IA e a sessão usados (reprodutibilidade).
- **A1 concluído** (feature spec-kit oficial `011-divulgacao-coautoria-ia`): nota de autoria adicionada à abertura (cap. 00, "Nota de autoria e método"), divulgando a co-autoria humano+IA sob responsabilidade humana, com ponteiro para o Guia §6. Os três achados ratificáveis (A1/A2/A3) do estudo estão agora incorporados.
- **IA (aplicação de A3):** agente **Claude Code (Anthropic)** sob curadoria/responsabilidade humanas; sessão registrada nos trailers de commit. *(O identificador interno do modelo é omitido dos artefatos por política de identidade da ferramenta; o autor humano pode anotá-lo à parte se desejar.)*

Edição 0.6 — 2026-07-26 · estudo de metodologias de escrita (ciclo spec-kit oficial)

- **Primeira feature pelo ciclo oficial do Spec Kit** (spec 010, branch `010-estudo-metodologias-escrita`): `/speckit-specify` → `/speckit-clarify` → `/speckit-plan` (com Constitution Check) → `/speckit-tasks` → `/speckit-analyze` → `/speckit-implement`, usando os scripts `.specify/` e os templates oficiais e seus gates — em contraste com as edições anteriores, que seguiram o *método* spec-driven mas escritas à mão.
- **Nova seção 6 do `GUIA-EDITORIAL.md`**: um *survey* das metodologias de escrita editorial e acadêmica — tradicionais (IMRaD, processo cognitivo, craft/estilo, argumento, peer review, design instrucional) e da era-IA (co-escrita, spec-driven, RAG/verificação, integridade/autoria, críticas) — com o **método deste livro declarado** e a **divulgação aberta de co-autoria humano+IA** (Claude Code sob responsabilidade humana), seguindo as políticas ICMJE/COPE/Nature/Science.
- **Bibliografia**: nova seção "Guia — Metodologias de escrita" com as fontes verificadas por busca cruzada.

Edição 0.5 — 2026-07-26 · visualizações React + unificação editorial v3

- **P2 concluída** (spec 001, branch `002-visualizacoes-react`): ilhas de visualização React no motor do livro — heatmap sortável do benchmark e registro de expiração com filtro, como *islands* (progressive enhancement; sem JS, ficam as tabelas Markdown). Fonte canônica em `benchmark/notas.json`.
- **Sete capítulos de funcionalidade trazidos ao esqueleto v3** (specs 003–009, um ciclo spec-kit por capítulo, branch `003-reescrita-editorial-v3`): 06 MCP, 08 Memória e Estado, 09 Planejamento, 10 Subagentes/Orquestração, 11 Verificação/Evals, 12 Extensibilidade, 13 Interfaces. Cada um ganhou objetivos de Bloom, **fundamentos científicos** (papers reais verificados por busca cruzada), **fontes da indústria** (docs de vendor/blogs), estado da arte no corpo, mão na massa, verificação e **Apêndice A** com as rodadas 2/frameworks.
- **Lacunas de bibliografia preenchidas/registradas**: o cap. 06 (MCP) saiu de "lacuna" para literatura de segurança consolidada (SoK, MCPTox, auditorias); os caps. 12 (extensibilidade) e 13 (interfaces) — sem canon *agent-specific* — foram ancorados em SE clássica e HCI, respectivamente, com a lacuna registrada honestamente (Princípio I).
- **Atualizações datadas (livro vivo)**: refutada a previsão de que "nenhum harness atua como *servidor* MCP no core" (rodada 2: Codex/Hermes/OpenClaw/OpenHands/n8n são cliente **e** servidor); o n8n **depreciou** seu Plan-and-Execute Agent (planejamento explícito recuando para trabalho longo); a verificação virou **adversarial** (reward hacking — o agente joga contra o verificador); e os formatos de extensão (SKILL.md/AGENTS.md) convergindo num padrão portável (o "MCP da extensibilidade").

Edição 0.4 — 2026-07-25 · publicação (feature 001, em andamento)

- **Primeira melhoria sob o Princípio VII** (spec-driven, branch `001-publicacao-latex-html`): spec → plan → tasks → implement.
- **Motor do livro próprio** (`publicar/`, Node): gera o site HTML navegável a partir do Markdown (`docs/`), com sidebar, navegação anterior/próximo, tema claro/escuro, selo de data de captura (livro vivo) e callouts pedagógicos. Fonte permanece Markdown; publicação é um adapter (portas-e-adaptadores). P1 concluída; P2 (viz React), P3 (PDF/LaTeX), P4 (CI + apêndice de infra) pendentes.

Edição 0.3 — 2026-07-25 · "livro vivo"

- Introduzido o sistema de datação (este arquivo, cabeçalhos de captura nos capítulos, o registro de expiração abaixo).
- Fase de edição v3 iniciada: capítulos 02, 03, 04, 05, 07 reescritos com "Fontes da indústria" + "Estado da arte" + "Apêndice A por repositório".
- harness-zero: endpoint gratuito NVIDIA NIM documentado.
- **Governança formalizada**: constituição do projeto preenchida (`.specify/memory/constitution.md`, v1.0.0) com os 6 princípios centrais — incluindo o framework pedagógico (princípio III) — e `CLAUDE.md` na raiz tornando-a a autoridade que todo trabalho deve seguir.

Edição 0.2 — 2026-07-25 · fundação pedagógica e camadas novas

- Parecer editorial, framework pedagógico (Backward Design + 4C/ID + Diátaxis + Carga Cognitiva), Guia Editorial.
- Capítulos novos: 15 (harness embutido), 16 (aprendizado auto-evolutivo), 17 (protocolos).
- harness-zero iniciado (etapas 0–1); bibliografia científica; spec-kit e skill academic-research.

Edição 0.1 — 2026-07-24 · fundação

- Introdução, fundamentos, 12 capítulos de dimensão, capítulo de convergências.
- Benchmark: rodada 1 (opencode, gemini-cli, OpenHarness), rodada 2 (Codex, Goose, Aider, OpenHands, OpenClaw, Hermes, IronClaw, n8n), rodada frameworks-1 (LangGraph, Agents SDK, CrewAI, software-agent-sdk); ohmo; retro dim-13.

Registro de expiração (o placar das previsões)

> A parte mais viva do livro. Cada componente de harness que descrevemos existe porque o modelo ainda não faz aquilo sozinho — e prevemos *quando* deixaria de ser necessário. Aqui pontuamos essas previsões contra a realidade, com data. É a única seção que **espera-se** que envelheça: quando uma linha vira "cumprida", o livro registrou a própria disciplina se dissolvendo em tempo real.

Estados: 🔵 aberta (prótese ainda necessária) · 🟡 em movimento (sinais de expiração) · 🟢 cumprida (o modelo/plataforma absorveu) · 🔴 refutada (a previsão estava errada; o componente é mais permanente do que pensávamos)

Componente	Existe porque…	Previmos que expira quando…	Estado	Evidência datada
Compactação (cap. 04)	janelas são finitas e caras	contexto longo ficar barato e confiável	🟡 em movimento	A compactação **mudou de dono** antes de expirar: Anthropic lançou compaction na API (beta `compact-2026-01-12`) e o Codex fez compactação remota v2 (2026). Não desapareceu — migrou do harness para a plataforma.
Prompt por família de modelo (cap. 03)	modelos respondem diferente a instruções	instruction-following convergir	🔵 aberta	Ainda divergente; Codex chegou a tornar o prompt server-driven por modelo (2026) — reforço, não expiração.
Plan mode imposto (cap. 09)	modelos agem precipitadamente	modelos planejarem sob risco espontaneamente	🔵 aberta	Planejamento seguiu como a dimensão mais fraca da indústria em todas as rodadas (2026-07); o n8n **depreciou** seu Plan-and-Execute Agent — o plano explícito recuou para trabalho longo/humano-no-loop, não expirou.
Policy engine / aprovações (cap. 07)	modelos não são confiáveis com ações destrutivas	confiabilidade calibrada e verificável	🔵 aberta	Consenso 2026: injection tratada como não-resolvível; esforço migrou para blast radius, não para confiar no modelo.
Verificação externa (cap. 11)	a auto-correção intrínseca não basta (o modelo não se conserta sozinho)	modelos verificarem o próprio trabalho de forma confiável	🔵 aberta	Reforçada, não expirando: "LLMs Cannot Self-Correct Reasoning Yet" (2310.01798) e o *reward hacking* (o agente apaga asserts/patcha o pytest) empurraram a indústria para verificador **externo e imutável** (testes held-out, verify-on-stop) — 2026-07.
Aprendizado auto-evolutivo (cap. 16)	— (cláusula invertida)	nunca — o harness *escreve* scaffolding em vez de esperar o modelo	🔵 aberta	Hermes e gemini-cli fecharam o ciclo (2026-07); é auto-expansão, não expiração.
Sandbox / contenção (cap. 07)	é sobre o mundo, não sobre o modelo	nunca (fronteira, não prótese)	🔴 não-expira	Confirmado nas 3 rodadas; contenção é o scaffolding que resta quando o modelo melhora.
Protocolos (MCP/A2A/ACP/AGENTS.md — cap. 17)	interoperabilidade entre sistemas	nunca (fronteira com o mundo)	🔴 não-expira	MCP, goose e AGENTS.md doados à **Agentic AI Foundation / Linux Foundation** (dez/2025); MCP em 10/11 harnesses e o ACP fundido no A2A sob a LF (ago/2025) — a fronteira se institucionaliza, não desaparece (2026-07).

Regra de manutenção: a cada rodada do benchmark e a cada edição, revisar esta tabela — promover 🔵→🟡→🟢 com a evidência datada que justifica. Uma linha que muda de estado é a notícia mais importante que uma nova edição pode trazer.

Guia Editorial — regras operacionais do livro

> Versão operacional das orientações pedagógicas. O parecer completo (com fundamentação) está em `estudos/2026-07-25-parecer-editorial-plano-pedagogico.md`. Este guia é o que se consulta **enquanto escreve**.

1. O framework pedagógico em quatro linhas

Framework	O que dita no livro
Backward Design	Todo capítulo se projeta de trás para frente: objetivos → evidências (verificação/prática) → só então o conteúdo
4C/ID	Etapas do harness-zero = tarefas inteiras; capítulos = informação de apoio; boxes no código = just-in-time; katas = treino de parte
Diátaxis	Quatro tipos de texto, nunca misturados na mesma seção: capítulo=explanation, harness-zero=tutorial, templates/benchmark=reference, "o que roubar"=how-to
Carga Cognitiva	Worked examples antes de exercício; exercícios são "complete", não "crie do zero"; andaime diminui etapa a etapa; uma ideia nova por vez

2. Esqueleto v3 de capítulo (obrigatório; piloto: cap. 04)

Regra de edição (v3): ao abrir cada tema, buscar também **material comercial/industrial** (docs oficiais de vendors, blogs de engenharia, posts de praticantes) além do científico. A fonte-base continua sendo **o código dos repositórios**. O corpo do capítulo recebe **o estado da arte** (o que está mais moderno, sintetizado de todas as rodadas do benchmark + indústria); o tratamento detalhado **por repositório vai para o Apêndice do arquivo** — que fica na versão online como complementação e é atualizado a cada rodada.

1. **Objetivos** — 3–5, verbos de Bloom (explicar, comparar, implementar, avaliar)
2. **O problema** — por que a dimensão existe
3. **Fundamentos científicos** — 2–4 papers *traduzidos para decisões*; ponteiro para `bibliografia.md`
4. **Fontes da indústria** — docs de vendor e posts de engenharia relevantes, com a mesma regra de tradução ("o vendor recomenda X porque Y")
5. **O estado da arte** — o corpo principal: padrões consolidados + o que há de mais moderno, citando repositórios apenas como exemplos nominais (o detalhe fica no apêndice)
6. **Mão na massa** — a etapa correspondente do harness-zero
7. **Síntese + "o que roubar"** — leitura executiva e ideias exportáveis
8. **Verificação** — 2–3 perguntas que testam exatamente os objetivos do item 1
9. **Apêndice A — Como cada repositório trata** — a evidência por harness com paths, expandida a cada rodada do benchmark (material de complementação online)

2.1 Livro vivo: datação e histórico (obrigatório)

Este é um **livro vivo** — coerência com a própria tese (a cláusula de expiração: o que descrevemos é temporário). Três regras:

1. **Todo capítulo v3 declara a data de captura no cabeçalho**: `> **Estado da arte capturado em AAAA-MM** · última revisão AAAA-MM-DD · [histórico](../HISTORICO.md)`. Isso diz ao leitor se a seção "Estado da arte" está fresca — o que a data do *evento* (no corpo) não faz.
2. **Distinguir três datas** (ver `HISTORICO.md`): data do evento (no corpo — fato histórico, imutável), data de captura (no cabeçalho — quando fotografamos), rodada do benchmark (nas avaliações — versão da foto de cada repo). Reavaliar = nova rodada, nunca sobrescrever.
3. **Toda edição atualiza `livro/HISTORICO.md`**: o changelog de edições, a tabela de snapshot por capítulo, e — o mais importante — o **registro de expiração** (o placar das previsões: cada cláusula de expiração pontuada 🔵/🟡/🟢/🔴 contra a realidade, com evidência datada). Uma linha que muda de estado é a notícia mais importante de uma nova edição.

Regra de escrita associada: quando uma afirmação for sensível ao tempo ("hoje", "ainda não", "o consenso de 2026"), ela está implicitamente sob a data de captura do cabeçalho — não precisa datar cada frase, mas evite absolutos atemporais ("nunca", "sempre") a menos que sejam do tipo não-expira (fronteira com o mundo).

3. Regras de escrita permanentes

- **Evidência por caminho de arquivo** para qualquer afirmação sobre um harness; **status ✓** para qualquer citação científica (skill `academic-research` tem o fluxo).
- Notas 0–3 só comparam dentro da mesma categoria do benchmark.
- Cada componente descrito deve, quando possível, declarar sua **cláusula de expiração**.
- Prosa em português; termos técnicos consagrados (harness, loop, tool, prompt) **sem tradução**.
- Tabelas para fatos enumeráveis; explicação vive na prosa, não nas células.

4. Regras do harness-zero (as 4 condições do parecer)

1. **DDD leve** — linguagem ubíqua = glossário do livro; padrão tático só onde paga; DDD aparece como consequência nomeada no código.
2. **Arquitetura por refatoração** — cada porta nasce da dor do capítulo correspondente; nunca estrutura antecipada.
3. **Anti-apodrecimento** — modelo atrás de `LLMPort`; etapas autocontidas e executáveis; erros didáticos deliberados são **comentados como tal** no código.
4. **Chat congelado** — HTML+JS servido pelo backend; só evolui quando uma dimensão exigir superfície nova.

5. Ferramentas do repositório

- **spec-kit** (`.specify/` + comandos `/speckit-*`): para features novas do harness-zero ou seções grandes do livro, o fluxo é `/speckit-specify` → `/speckit-plan` → `/speckit-tasks` → `/speckit-implement` (com `/speckit-clarify` antes do plano quando o pedido for ambíguo). A constitution do projeto vive em `.specify/memory/`.
- **Skill `academic-research`** (`.claude/skills/`): fluxo de localizar → validar → registrar → integrar referências científicas.
- **`scripts/sync-forks.ps1`**: sincronização local dos forks com upstreams.

6. Estudo: processos e metodologias de escrita editorial e acadêmica (tradicionais e da era-IA)

> **Atualizado em 2026-07** · livro vivo (as práticas de IA têm data de expiração). Fontes na seção "Guia — Metodologias de escrita" de `bibliografia.md`.

Um livro sobre engenharia — a disciplina de instrumentar bem um processo — precisa expor o próprio processo de produção, ou contradiz o que ensina. Esta seção é um *survey* das metodologias de escrita editorial e acadêmica (as consagradas e as da era-IA) e, ao fim, torna explícito e datado o método com que este livro é escrito. É texto de **referência/explicação** (Diátaxis), não um capítulo — por isso não segue o esqueleto v3.

6.A — Metodologias tradicionais

Estrutura da escrita científica. O **IMRaD** (Introdução, Métodos, Resultados, Discussão) não foi inventado por um autor: Sollaci & Pereira (2004) mostram que foi "imposto por decantação", virando padrão nos anos 1980. O clássico Gopen & Swan, "The Science of Scientific Writing" (1990) estabelece o princípio da *expectativa do leitor* — o sentido nasce da posição estrutural (topic/stress positions), não só das palavras. A codificação prática está em *How to Write and Publish a Scientific Paper* (Day & Gastel).

A escrita como processo cognitivo. Flower & Hayes (1981) modelam a escrita como processos **recursivos** (planejar/traduzir/revisar) guiados por objetivos, não etapas lineares; Sommers (1980) mostra que escritores experientes revisam *re-vendo o sentido*, enquanto novatos trocam palavras na superfície — "escrever é reescrever".

Craft e estilo. A tradição vai do minimalismo prescritivo de *The Elements of Style* (Strunk & White) à teoria *principiada* da clareza de *Style: Toward Clarity and Grace* (Williams — personagens=sujeitos, ações=verbos, velho-antes-de-novo), passando pela voz autêntica de *On Writing Well* (Zinsser); os padrões editoriais/citação são o *Chicago Manual of Style* (17ª ed.) e o *APA Publication Manual* (7ª ed.).

Craft of research e argumento. *The Craft of Research* (Booth, Colomb & Williams) enquadra pesquisa como **fazer um argumento a um leitor** (problema → pergunta → *claim* → razões → evidência; o "So what?"); o modelo de Toulmin (1958) dá a anatomia do argumento (claim, grounds, warrant, backing, qualifier, rebuttal).

Revisão por pares e fluxo editorial. Spier (2002) traça a história do peer review; a historiografia (Baldwin, ETHOS) lembra que o refereeing universal é construto do séc. XX. E a divisão de trabalho editorial — *developmental editing* (reestruturar visão/discurso) × *copyediting* (preparo de sentença) — é o eixo do fluxo (Norton, *Developmental Editing*).

Design instrucional (o que este livro já usa). Backward Design (Wiggins & McTighe), 4C/ID (van Merriënboer et al., 2002), carga cognitiva (Sweller, 1988) e Diátaxis (Procida) — a base pedagógica do Princípio III.

6.B — Metodologias da era-IA

Co-escrita humano-IA. Estudos de HCI tratam a co-escrita como interação **observável**, não caixa-preta: CoAuthor (Lee, Liang, Yang, 2022) registra a interação em nível de keystroke; Wordcraft (Yuan et al., 2022) decompõe a escrita em *moves* (continuar/infill/elaborar/reescrever) ligados à intenção. **Cautela** medida: Jakesch et al. (2023) mostram que um assistente enviesado desloca o que o usuário escreve *e pensa* ("persuasão latente").

Spec-driven / structured authoring / docs-as-code. Escrever a intenção primeiro e deixá-la dirigir a geração: GitHub Spec Kit (spec → plan → tasks → implement) e Amazon Kiro formalizam isso; a comunidade de documentação já adota workflow de engenharia para prosa (docs-as-code, SIGDOC '24; DITA/topic-based).

Pesquisa aumentada por agentes e recuperação. RAG (Lewis et al., 2020) ancora a geração em fontes recuperadas em vez da memória do modelo; a fronteira agêntica decompõe o *survey* em papéis (buscar/sintetizar/verificar) — tendência ilustrada por trabalhos de auto-survey (⏳ a confirmar).

Verificação e proveniência. RARR (Gao et al., 2023) faz atribuição/checagem *após* a geração; Liu, Zhang & Liang (2023) medem que só **51,5%** das afirmações de motores de busca generativos são totalmente suportadas por citação — julgar por *citation precision/recall*; watermarking (Kirchenbauer et al., 2023) embute proveniência (frágil a paráfrase).

Integridade acadêmica e autoria. O consenso das políticas: **um LLM não pode ser autor** (não responde pelo conteúdo) e o uso deve ser **divulgado** — ICMJE, COPE (2023), *Science* (Thorp, 2023), *Nature* (2023). E a divulgação, na prática, é amplamente violada (Academ-AI, 2024).

6.C — Tensões e síntese (tradicional × IA)

O ganho da assistência de IA (velocidade, alcance de pesquisa, estrutura) vem com quatro tensões que uma edição acadêmica não pode ignorar:

- **Fontes fabricadas.** Walters & Wilder (2023) mediram **55%** de citações fabricadas no GPT-3.5 (18% no GPT-4) e erros substantivos nas reais — daí a regra deste livro: **verificar toda referência** contra a fonte primária, por busca cruzada.
- **Verifiabilidade.** Texto que *parece* citado frequentemente não é suportado (os 51,5% de Liu et al.) — a citação precisa ser conferida, não confiada.
- **Reprodutibilidade.** Saídas de LLM são não-determinísticas; logar prompt, versão de modelo e contexto é parte do rigor.
- **Homogeneização e "cognitive debt".** A IA converge estilo e ideias (homogeneização, 2024) e o uso acrítico associa-se a menor engajamento/propriedade (Kosmyna et al., 2025) — razão para a IA *ampliar*, não *substituir*, o julgamento do autor.

A síntese do livro: usar a IA como **prótese de pesquisa e estruturação sob verificação humana**, não como autor. As metodologias tradicionais (argumento, clareza, revisão) permanecem o padrão de qualidade; as de IA aceleram o caminho até ele, desde que cercadas de verificação.

6.D — O método deste livro, declarado

Este livro pratica o que descreve. Cada prática liga-se a um princípio da constituição e tem evidência no próprio repositório:

- **Evidência acima de retórica** (Princ. I) — nenhuma afirmação sobre um harness sem *path* no código; nenhuma citação sem status validado. Fontes verificadas por busca cruzada; lacunas registradas, não preenchidas com fonte fraca.
- **A fonte-base é o código** (Princ. II) — o corpo nasce da leitura do código dos harnesses; ciência e indústria contextualizam. O tratamento por repositório (com paths) é o **Apêndice A** de cada capítulo.
- **Método pedagógico combinado** (Princ. III) — Backward Design + 4C/ID + Diátaxis + carga cognitiva; o esqueleto v3 é a materialização.
- **Pesquisa dupla verificada** — ao abrir cada tema, agentes de pesquisa em paralelo levantam material **científico** e **de indústria**; cada fonte é confirmada por ≥2 menções independentes antes de entrar (a regra que a era-IA torna ao mesmo tempo possível e obrigatória, à luz de Walters & Wilder).
- **Ciclo spec-driven** (Princ. VII) — toda melhoria passa por `spec → plan → tasks → implement` (spec-kit), em branch própria; *esta seção* foi produzida assim (`specs/010-estudo-metodologias-escrita/`), com o ciclo oficial e seus gates (Constitution Check, análise cross-artefato).
- **Livro vivo** (Princ. IV) — datação e `HISTORICO.md`; as previsões têm um placar (registro de expiração).

Divulgação de autoria (transparência). Coerente com as políticas acima e com o Princípio I, declaramos abertamente: este livro é **co-escrito com um agente de IA (Claude Code, da Anthropic)** operando sob **autoria, curadoria e responsabilidade humanas**. O agente executa pesquisa, redação e o ciclo spec-kit; o autor humano define o escopo, decide (via `/speckit-clarify` e revisão), verifica as fontes e responde pelo conteúdo. Seguindo ICMJE/COPE/*Nature*/*Science*, a IA **não** é listada como autora — não pode ser responsável — e seu uso é divulgado aqui, no método.

6.E — Fluxo repetível para um contribuidor

Para levar um capítulo ou seção ao padrão do livro:

1. **Abrir o tema** — pesquisa dupla (comercial/industrial + científica), verificada por busca cruzada; registrar lacunas.
2. **Reunir a fonte-base** — ler o código dos harnesses; anotar paths (vira Apêndice A).
3. **Escrever** — no esqueleto v3 (capítulos) ou no tipo Diátaxis correto (guia/benchmark = referência); um tipo de texto por seção; termos técnicos sem tradução.
4. **Revisar (developmental)** — re-ver estrutura e sentido antes do copyedit de superfície: o argumento fecha? a ordem serve ao leitor? há redundância ou lacuna? "Escrever é reescrever" (§6.A; portão de qualidade da constituição).
5. **Verificar fontes** — nenhuma URL/ID inventado; não-confirmado marcado ⏳; sincronizar `bibliografia.md`.
6. **Gate de build** — `node publicar/build.mjs` verde (sem link interno quebrado).
7. **Datar** — selo de captura no capítulo e entrada no `HISTORICO.md` — **com a versão do modelo de IA usada** — se o estado da arte mudou.

Salvaguardas de uso de IA: a IA pesquisa e rascunha; o humano decide, verifica e assina. Toda fonte trazida por um agente é conferida antes de entrar no corpo.

Siglas e glossário (política)

- **Toda sigla técnica é apresentada por extenso na 1ª ocorrência** de um capítulo — "Model Context Protocol (MCP)" — e, dali em diante, o texto pode usar só a sigla.
- O motor de publicação reforça isso: **envolve automaticamente cada sigla conhecida em `<abbr>`**, de modo que passar o mouse revela o significado em qualquer ocorrência, sem poluir o texto-fonte. O mapa de siglas vive em `publicar/build.mjs` e é espelhado na página **Glossário** (`livro/glossario.md`).
- O **Glossário** dá o **por extenso**, uma explicação curta e **em que capítulos** cada sigla aparece. Ao introduzir uma sigla nova, adicione-a nos dois lugares (mapa do motor + glossário) e **confira a expansão na fonte** (Princípio I).

Cadência do livro vivo

> Política decidida no ADR 0007 (alternativas e justificativa lá).

- **Janela trimestral** (próxima: **2026-10**): re-sync dos 16 forks (`scripts/sync-forks.ps1`), diff dirigido pelas dimensões do benchmark, atualização dos Apêndices A afetados, do placar de expiração e das datas de revisão; edição minor no Histórico.
- **Gatilho extraordinário**: qualquer evento que **invalide uma "Leitura executiva"** (mudança de protocolo, capacidade migrando para o provedor, harness do corpus arquivado) dispara revisão pontual do capítulo afetado, sem esperar a janela.
- A data "estado da arte capturado em" de cada capítulo continua sendo a verdade exposta ao leitor — a cadência existe para que ela nunca minta por omissão.

Sobre

Sobre o autor

Gilsiley Henrique Darú é o editor, direcionador e orquestrador humano deste livro vivo. Cientista de dados, engenheiro e professor universitário, atua há mais de 20 anos na fronteira entre **otimização, pesquisa operacional e inteligência artificial** aplicadas ao planejamento e à cadeia de suprimentos — e há alguns anos, na engenharia dos sistemas de IA que este livro chama de *harness*.

A escolha do tema não é acidental. A trajetória do autor é uma longa prática de **envolver métodos poderosos em um andaime que os torna úteis no mundo real**: solvers matemáticos dentro de rotinas de PCP, modelos preditivos dentro de processos de negócio, e agora agentes de IA dentro de um *scaffolding* de loop, ferramentas, memória e verificação. Este livro é a sistematização dessa disciplina.

> Perfis: Currículo Lattes · ORCID 0000-0002-8979-0461 · LinkedIn · Contato: ghdaru@gmail.com

Em uma frase

Head de Dados & IA, consultor de otimização empresarial, IA e supply chain — aplicando **Teoria das Restrições**, pesquisa operacional, pensamento sistêmico e agentes de IA para melhorar fluxos e encontrar foco. Baseado em Joinville, Santa Catarina.

Formação acadêmica

- **Doutorado em Métodos Numéricos em Engenharia (Matemática Computacional)** — Universidade Federal do Paraná (UFPR), em andamento (desde 2019). Tese em desenvolvimento: *Framework Humano-Computacional para Saneamento Acelerado de Dados*.
- **Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria (Ciência de Dados)** — Universidade de São Paulo (USP), 2021–2024. Dissertação: *Categorização de produtos em e-commerce: avaliação do método Argmax para classificação de descrições curtas em português*.

- **Mestrado em Métodos Numéricos em Engenharia** — UFPR, 2003–2005. Dissertação: *Uma heurística para o sequenciamento da produção baseada na Teoria das Restrições* (programação matemática, otimização, scheduling e heurísticas).
- **Especialização em Ciência de Dados (SPRINT)** — Business Intelligence, Inteligência Artificial e Big Data — SENAI/DR-SC, Florianópolis, 2018–2019.
- **Especialização em Engenharia de Software com ênfase em Orientação a Objetos** — Pontifícia Universidade Católica do Paraná (PUC-PR), 2004.
- **Engenharia Mecânica** — Universidade do Estado de Santa Catarina (UDESC).
- **Tecnólogo em Processamento de Dados** — UDESC.

Atuação profissional

Neogrid (2016–presente) — empresa de software referência em integração da cadeia de suprimentos. Trajetória de responsabilidade crescente, do NGLabs à liderança de dados e IA:

- **Head de Dados & IA** (desde dez. 2025) — engenharia de dados (captura, ingestão, transformação, disponibilização), governança (dados mestres, referência, metadados, qualidade), ciência de dados e analytics (análises descritivas, diagnósticas, preditivas e prescritivas) e direcionamento das iniciativas de IA, incluindo a construção de um hub contextual e agêntico.
- **Gerente Executivo de Inovação** (jul.–dez. 2025).
- **Gerente do Laboratório de Inovação e IA em Supply Chain Management** (2023–2025) — prototipagem de soluções, experimentação orientada a dados e desenvolvimento do time em análise avançada.
- **Especialista / Cientista de Dados** e **Coordenador de Qualidade de Dados** (2016–2023), no NGLabs — coleta, limpeza, transformação, mineração e comunicação de conhecimento a partir de dados.

Como **consultor de otimização e supply chain**, atendeu grandes operações — varejo (HAVAN), otimização ferroviária na Alemanha (HVLE) e planejamento de produção na moda (Malwee) — aplicando Teoria das Restrições, pesquisa operacional e simulação discreta e contínua.

Experiência anterior na indústria (mais de 20 anos no total):

- **Grupo Malwee** (2014–2016) — Gestor de PCP e Coordenador Técnico Executivo: melhoria da capacidade de planejamento, redução de estoques, cálculo e categorização de estoque de segurança, plano mestre e apoio à migração para SAP (módulo PP/MRP).
- **WEG Automação** (2006–2014) — Coordenador de PCP e Analista de Sistemas: implantação de S&OP, políticas de estoque com aumento de OTIF, aprimoramento da previsão de vendas, plano mestre, indicadores de performance e migração SAP (com projeto de APO).
- **Datasul** (2000–2005) — Analista de Sistemas e de Negócios: desenvolvimento de aplicações com pesquisa operacional, programação por restrições, algoritmos genéticos e mineração de dados (corte de bobinas, alocação de pessoal, petroquímica).

Docência

Professor universitário há mais de duas décadas, com passagem por coordenação de curso e pós-graduação:

- **Universidade do Estado de Santa Catarina (UDESC)** — Professor universitário (desde 2017): Gestão da Produção, Gestão Empresarial, Empreendedorismo e Pesquisa Operacional.
- **Centro Universitário — Católica de Santa Catarina** — Professor de **Pós-Graduação em IA & Deep Learning** (2024–2025); Pesquisa Operacional I e II na graduação em Engenharia de Produção.
- **INESA — Instituto de Ensino Superior Santo Antônio** — Professor (2017–2022): Estatística e Matemática Aplicada; participou da criação do PPC e da aprovação do curso de Engenharia de Produção.
- **SOCIESC** — Professor titular (2011–2019): Linguagens Formais e Autômatos, Análise de Algoritmos, Compiladores, Teoria da Computação, Cálculo III, Estrutura de Dados.
- **Faculdade Metropolitana de Guaramirim (FAMEG) — Coordenador do curso de Engenharia de Produção** (2011–2014) e professor (2009–2011): Pesquisa Operacional, Análise Multicritério, Modelagem e Simulação, Estatística, Cálculo. Atuação em ENADE, autorização e reconhecimento de curso.
- Também lecionou em FACASC, Centro Universitário Católica de Santa Catarina (Jaraguá do Sul), Faculdade Cenecista de Joinville e Associação Catarinense de Ensino.

Produção acadêmica

Artigos completos em periódicos

- Bianchini, J.; **Darú, G. H.**; Berger, S. *Analysis of production planning and control based on the simulation of a production process using a hybrid MRP and Kanban model*. **Journal of Lean Systems**, v. 3, n. 1, 2018. [artigo]
- Bratti, R. T.; Coelho, E. T. B.; **Darú, G. H.**; Decker, S. L.; Steffen, D. K. *A influência da lei 9870/99 sobre a inadimplência no setor educacional em uma instituição de ensino de Santa Catarina*. **Paidós**, v. 16, 2019.
- Bratti, R. T.; Coelho, E. T. B.; **Darú, G. H.**; Decker, S. L.; Reis, E. A. *A Teoria do Crescimento da Firma e Fusões e Aquisições: evidências da inter-relação das teorias*. **Paidós**, v. 16, 2019.
- Bratti, R. T.; **Darú, G. H.**; Machado, C. S. G.; Pavanati, I.; Reis, E. C. S. *O ensino da matemática através de jogos na perspectiva da neurociência com alunos do 3º ano do Ensino Fundamental*. **Paidós**, v. 16, 2019.

Trabalhos em anais

- **Darú, G. H.**; Pimentel, R.; Maldonado, M. U. *Manufactured fabrics obsolescence risk evaluation of alternative policies in a textile industry*. **XIV CLADS — Congresso Latino-Americano de Dinâmica de Sistemas**, São Paulo, 2016.

Orientações concluídas (seleção)

Trabalhos de conclusão em Engenharia de Produção e Computação, entre eles: aplicação de **algoritmos genéticos** para produção de cortes de barras; comparação entre **busca semântica e busca textual**; avaliação de cenários produtivos por **simulação**; modelos de **apoio multicritério à decisão** (PROMÉTHÉE) para seleção de fornecedores; e sistemas de apoio à decisão com **programação por restrições**.

Como citar este livro

Esta obra tem **DOI** (Zenodo/DataCite) e é versionada por edição:

> Darú, Gilsiley Henrique. *Engenharia de Harness — Um livro vivo sobre o scaffolding que envolve agentes de IA*. 2026. DOI: 10.5281/zenodo.21632412

O identificador acompanha a obra viva; cada edição também recebe seu próprio DOI de versão. A co-autoria humano + IA está declarada na Nota de autoria e método.

Perfis e contato

- **Currículo Lattes**: https://lattes.cnpq.br/6253911800847523
- **ORCID**: https://orcid.org/0000-0002-8979-0461
- **LinkedIn**: https://www.linkedin.com/in/gilsiley-darú/
- **E-mail**: ghdaru@gmail.com

Esta página é parte do aparato do livro (back matter). Os fatos vêm do Currículo Lattes, do perfil profissional público e de fontes verificáveis; instituições e empresas são citadas como trajetória, não como endosso. Sobre a co-autoria humano + IA desta obra, veja a Nota de autoria e método na introdução.