

ASP: uma *skill* operacional de pré-mortem para agentes de código com LLM

Um relato de experiência

Carlos Ulisses Flores

Codex Hash Research Laboratory

ORCID: 0000-0002-6034-7765

Codex Hash Research Report 2026-01

Data: 17 de novembro de 2026

Resumo

Assistentes de código baseados em LLM costumam produzir código *localmente correto* que omite modos de falha que um engenheiro sênior levantaria num pré-mortem, e a produção quebra de formas caras. Chamamos esses itens ausentes de *shadow points*. Este relato descreve o Anticipating Shadow Points (ASP), um protocolo de planejamento em 13 fases distribuído como *skill* do Claude Code, que integra o pré-mortem de Klein (2007) à taxonomia de falhas multiagente de Cemri et al. (2025), confere o plano com um validador que não compartilha o contexto do planejador, conforme a literatura sobre limites da autocorreção (Huang et al., 2024; Tyen et al., 2024), e, num modo opcional, entrega a execução a um subprocesso filho com contrato em arquivo. Documentamos o desenho, a implantação como *plugin* de código aberto e observações preliminares de cinco avaliações de engenharia, lidas pelo autor sem cegamento nem transcrições preservadas. Um estudo pré-registrado mediu o protocolo em seis repositórios públicos: 90 rodadas com o Claude Opus 5.5, riscos de referência listados às cegas por três modelos de outros fornecedores e dois juízes cegos. O ASP cobriu 48% dos riscos de referência, contra 66% do Claude Code simples com o pedido de listar, antes de planejar, o que pode dar errado. Ficou atrás nas seis tarefas: nenhum benefício mensurável pela regra fixada antes. Esse pedido elevou a cobertura sobre um pedido simples de planejamento de 56% para 66% (p = ,001), uma comparação secundária pré-registrada que um estudo confirmatório, relatado à parte, já testou como pergunta principal.

Palavras-chave: agentes de código LLM; pré-mortem; modos de falha multiagente; Claude Code; protocolos operacionais; pré-registro

Abstract

LLM coding assistants often produce *locally correct* code that omits failure modes a senior engineer would surface in a pre-mortem, and production then breaks in ways that are expensive to debug. We call these missing items *shadow points*. This experience report describes Anticipating Shadow Points (ASP), a 13-phase planning protocol shipped as a Claude Code skill that integrates Klein's (2007) pre-mortem with the multi-agent failure taxonomy of Cemri et al. (2025), checks the plan with a validator that does not share the planner's context, following the self-correction-limit literature (Huang et al., 2024; Tyen et al., 2024), and, in an opt-in mode, hands execution to a child subprocess with a file-based contract. We document the design, its deployment as an open-source plugin, and preliminary observations from five engineering evaluations, read by the author without blinding and with transcripts not preserved. A pre-registered study then measured the protocol on six public repositories: 90 trials on Claude Opus 5.5, reference risks listed blind by three models of other vendors, and two blind judges. ASP covered 48% of the reference risks against 66% for plain Claude Code asked to list the ways the change could go wrong before planning. It was behind on all six tasks: no measurable benefit under the rule fixed in advance. That single request raised coverage over a plain planning request from 56% to 66% ($p = .001$), a pre-registered secondary comparison that a confirmatory study, reported separately, has since tested as its primary question.

Keywords: LLM coding agents; pre-mortem; multi-agent failure modes; Claude Code; operational protocols; pre-registration

Sumário

1. Introdução 5
1.1. Contribuições 7
2. Fundamentação 8
2.1. Pré-mortem e retrospecção prospectiva 8
2.2. Modos de falha multiagente (MAST) 9
2.3. Limites da autocorreção e isolamento do validador 10
2.4. Trabalho relacionado mais próximo: o PreFlect 11
2.5. Linhagem clássica de engenharia de software e de qualidade 12
3. O Protocolo ASP 17
3.1. Visão geral 17
3.2. Posicionamento arquitetural 18
3.3. Iron Laws 20
3.4. O contrato de micro-TODO 21
4. Implantação Operacional 21
5. Observações Preliminares 23
5.1. O conjunto de avaliações 23
5.2. O que foi observado em execuções reais (autor, não cego) 24
5.3. O que o adjudicador determinístico mostra sobre si mesmo 25
5.4. O achado do código de saída do claude -p 28
5.5. Uma medição pré-registrada (2026-11) 30
6. Hipóteses de Projeto 34
7. Ameaças à Validade 36
8. Limitações e Trabalhos Futuros 38
9. Conclusão 39
10. Declarações 41
Agradecimentos 42
Referências 43
Apêndice A — Situação da verificação das citações 49
Apêndice B — Conjunto de avaliações e reprodutibilidade 51
Nota Técnica 54
Sobre o Autor 56

1. Introdução

Na prática do autor, trabalho de engenharia rotineiro — adicionar uma coluna a um banco de dados de produção, implantar uma função que chama uma API externa, refatorar um utilitário usado em trinta arquivos — é hoje muitas vezes entregue a assistentes de código com IA. Quando o assistente acerta, o dia do engenheiro fica livre. Quando erra de forma *sutil* — escrevendo código que compila, passa nos próprios testes, vai para produção, mas omite em silêncio um passo que um engenheiro sênior teria apontado de antemão —, a produção quebra de formas caras de depurar. A ciência da decisão tem uma técnica para exatamente essa lacuna, o pré-mortem (Klein, 2007), e a pesquisa em agentes agora dispõe de um catálogo de como sistemas multiagente de LLM falham (Cemri et al., 2025). Nossa tese é que as duas pertencem ao mesmo lugar em tempo de execução: antes de escrever código, um agente deve ser levado a fazer um pré-mortem contra uma taxonomia de falhas fixa, e seu plano deve ser conferido por um validador que não enxerga o raciocínio do planejador.

O modo de falha dos agentes de código com LLM que motivou o ASP não é o código *ruim*. É o código *incompleto*: o agente faz o que foi pedido, mas não o que alguém com profundo contexto de domínio teria se perguntado *antes de começar*. O atraso de réplica durante uma migração de esquema. O cabeçalho CORS na *edge function*, que diz ao navegador quais outros sites podem chamá-la. A política de segurança por linha (RLS), que limita as linhas que cada usuário pode ler e precisa ser atualizada para acompanhar a coluna nova. Chamamos esses itens ausentes de **shadow points** — modos de falha óbvios para um profissional que já se queimou com eles, invisíveis para um agente que não se queimou.

Duas vertentes da literatura convergem nessa lacuna. A primeira, na ciência da decisão, é o **pré-mortem**: um exercício estruturado em que uma equipe imagina que um projeto já fracassou e

enumera as causas antes de agir. A técnica foi operacionalizada para equipes de projeto por Klein (2007) e fundamentada no trabalho empírico anterior de Mitchell, Russo e Pennington (1989) sobre retrospecção prospectiva (*prospective hindsight*); Klein (2007) a resume como um aumento de cerca de 30% na capacidade de identificar razões para um resultado futuro, número que não conseguimos recalcular a partir do artigo de 1989 (Apêndice A). A segunda, na pesquisa em agentes, é a caracterização recente dos modos de falha multiagente por Cemri et al. (2025), que derivaram uma taxonomia de 14 modos, a Multi-Agent System Failure Taxonomy (MAST), a partir de 150 traços iniciais de sistemas multiagente de LLM (concordância entre anotadores κ = ,88) e liberaram um corpus de mais de 1.600 traços anotados. A crítica do plano de um agente antes de ele agir já existe (Seção 2.4); o que não encontramos foram as duas vertentes integradas como uma **disciplina de tempo de execução**, com artefatos nomeados e um validador que não compartilha o contexto do planejador, que um agente de código possa ser levado a seguir em toda tarefa não trivial.

Este artigo relata o **Anticipating Shadow Points (ASP)**, um protocolo de planejamento em 13 fases distribuído como *skill* do Claude Code, que combina o raciocínio de pré-mortem com a lista de verificação de 14 modos do MAST, separa planejamento de validação sob isolamento de *prompt* e, em seu modo opcional --goal, entrega a execução a um subprocesso filho com contrato em arquivo (por padrão, a execução fica na sessão do usuário). Implantamos a *skill*, distribuímo-la por três caminhos de instalação (*marketplace* de *plugins*, modo de desenvolvimento, *script* autônomo), submetemo-la a cinco avaliações de engenharia e relatamos o desenho e o que as avaliações podem e não podem mostrar. Essas cinco avaliações são preliminares e não medidas: GREEN (o agente com a disciplina do ASP) em cinco tarefas e RED (instruído a planejar sem ela) em três, uma execução cada, um único modelo, lidas pelo autor sem cegamento, com as transcrições brutas não

preservadas. Relatamos essas avaliações como observação qualitativa. Uma medição pré-registrada veio depois (Seção 5.5): em seis tarefas, as fases de planejamento do ASP cobriram menos riscos de referência do que um pedido de uma frase. As quatro escolhas de projeto que tratamos como hipóteses, e uma quinta medida no artigo complementar, estão na Seção 6.

1.1. Contribuições

Este relato de experiência contribui com:

1. **O protocolo ASP como projeto documentado** — treze fases, cada uma emitindo um artefato nomeado, organizadas em torno do princípio de que nenhuma implementação começa antes de os shadow points terem sido levantados.
2. **Doze restrições operacionais a que chamamos de "Iron Laws"** (Seção 3.3), cada uma ligada a uma fonte primária da literatura ou a uma decisão de projeto documentada. Duas têm uma verificação em nível de *shell* no *template* do *runner* que o agente é instruído a executar (a saída JSON da Iron Law 12, um padrão do *template* que o operador pode sobrescrever, e a *flag* ASP_IN_GOAL da Iron Law 9, definida para o filho para que o ASP não recomece dentro dele, cuja verificação na entrada é um trecho de Bash que a *skill* manda o agente executar); as outras dez são instruções em nível de *prompt*.
3. **Observações preliminares e qualitativas** de cinco avaliações de engenharia, e um adjudicador determinístico com proveniência criptográfica, exercitado em *fixtures* construídos (planos de exemplo que o autor escreveu para testá-lo) e pronto para pontuar transcrições reais sem confiar na adjudicação do autor (as listas esperadas continuam sendo do autor, e o *sidecar* opcional de sinônimos não é fornecido para nenhuma tarefa).

4. **Um achado empírico separado** sobre a semântica do código de saída do claude -p (o Claude Code executado de modo não interativo: um *prompt* entra, um resultado sai) sob recusa educada do agente, caracterizado no artigo complementar Flores (2026b). O achado motivou duas das Iron Laws, mas merece caracterização independente com N maior.

Este é um **relato de experiência** no sentido de sistemas, não um artigo de *benchmark*. Documentamos um sistema que construímos, uma metodologia que implantamos e o que observamos; não alegamos uma contribuição empírica robusta à literatura de *benchmarks* de agentes de LLM. A Seção 7 enumera em detalhe as ameaças à validade.

2. Fundamentação

Revisamos quatro vertentes de trabalhos recentes que informam diretamente o desenho do ASP (§§2.1–2.4) e depois situamos o ASP entre as disciplinas mais antigas de engenharia de software e de qualidade que ele transpõe (§2.5). Mantemos a revisão dos trabalhos recentes curta em relação ao habitual nos artigos de sistemas de agentes, porque a contribuição aqui é operacional e o contexto da literatura está a serviço dela.

2.1. Pré-mortem e retrospecção prospectiva

A técnica do pré-mortem pede a uma equipe que imagine, antes do compromisso, que um projeto já fracassou de forma catastrófica, e que enumere as causas. A base empírica é Mitchell, Russo e Pennington (1989), que mostraram que imaginar um evento futuro como certo (e não como incerto) produz explicações mais longas e mais episódicas; Klein (2007) resume o efeito como um ganho de cerca de 30% na identificação correta de razões para um resultado futuro. Klein (2007) operacionalizou o achado como o “pré-mortem” de equipes de projeto. Sua base experimental é

tênue: não encontramos replicação do efeito de cerca de 30% que Klein (2007) atribui a Mitchell et al. (1989), nem metanálise dele, no Crossref, no arXiv nem numa busca geral na web (novembro de 2026). Isso é uma ausência nesses índices, não prova de que nenhuma exista, e nada diz sobre experimentos com outros efeitos do pré-mortem. Normas de avaliação de risco catalogam técnicas vizinhas — a análise de cenários e a técnica estruturada do "e se" (SWIFT) —, mas o sumário da IEC 31010:2019, Anexo B, não lista o pré-mortem pelo nome. O termo *prospective hindsight* também foi reaproveitado para um sinal de aprendizado por reforço sem relação com ele (J. Zhang et al., 2026); nós o usamos apenas no sentido de Mitchell et al. A crítica do plano de um agente antes de ele agir já existe (PreFlect; Wang et al., 2026; §2.4). O que não encontramos é a integração que o ASP faz: um pré-mortem conduzido contra uma taxonomia de falhas fixa, registrado como artefatos nomeados e revisado por um validador que não compartilha o contexto do planejador.

2.2. Modos de falha multiagente (MAST)

Agentes de LLM intercalam raciocínio e ação num único ciclo (Yao et al., 2023) e são cada vez mais compostos em fluxos de trabalho multiagente cujos modos de falha se acumulam. Cemri et al. (2025) introduzem o **MAST** com o **MAST-Data**, um corpus de mais de 1.600 traços anotados de sete *frameworks* multiagente populares, como o AG2 (antigo AutoGen; Wu et al., 2023). A partir de uma análise rigorosa de 150 traços iniciais (concordância entre anotadores κ = ,88), derivaram quatorze modos de falha organizados em três categorias: especificação e projeto do sistema (5 modos), desalinhamento entre agentes (6 modos) e verificação e término de tarefas (3 modos); as tabelas da Seção 3 citam esses modos pelos seus códigos, de FM-1.1 a FM-3.3. A lista de 14 modos é a fundamentação empírica da lista de verificação de modos de falha do ASP: o ASP adota os nomes dos modos literalmente e operacionaliza cada um num *prompt* de pré-mortem mais uma mitigação, nos arquivos da *skill* complementar. Dois trabalhos concorrentes estendem a

direção da taxonomia: Zhu et al. (2025) introduzem o AgentDebug e a AgentErrorTaxonomy (memória / reflexão / planejamento / ação / nível de sistema), e S. Zhang et al. (2025), um Spotlight do ICML 2025, abre a **atribuição automatizada de falhas** com o conjunto de dados Who&When. O MAST é, portanto, uma taxonomia entre várias: o TRAIL formaliza uma taxonomia de erros agênticos sobre 148 traços anotados por humanos (Deshpande et al., 2025), o AgentRx anota 170 trajetórias fracassadas com um passo crítico e uma categoria de uma taxonomia de teoria fundamentada (Barke et al., 2026), e o AgenTracer relata que os LLMs de raciocínio atuais atribuem falhas em traços multiagente com acurácia geralmente abaixo de 10% (G. Zhang et al., 2025). O ASP usa o MAST como uma biblioteca de *prompts* para antecipar falhas, não como classificador das observadas, de modo que a escolha da taxonomia delimita o que a Fase 2 pergunta, e não o que ela consegue detectar.

2.3. Limites da autocorreção e isolamento do validador

O argumento para separar o validador do planejador se apoia em três trabalhos primários segundo os quais os LLMs não são confiáveis ao criticar a si mesmos sob contexto compartilhado. Esses achados motivam, mas não estabelecem, a hipótese de que separar o *prompt* do validador da transcrição do planejador melhora a validação no ASP. Huang et al. (2024, ICLR) mostram que a autocorreção intrínseca, sem retroalimentação externa, muitas vezes degrada o desempenho. Tyen et al. (2024) decompõem a autocorreção em localização e correção e mostram empiricamente que os LLMs não conseguem *localizar* os próprios erros de raciocínio, mas conseguem corrigi-los quando a localização é dada externamente — *a localização é o gargalo*. Zheng et al. (2023, NeurIPS D&B) documentam que juízes LLM fortes favorecem em excesso as saídas de modelos da mesma família (um "viés de autoaprimoramento"), mesmo ao julgar objetivamente a qualidade. Kamoi et al. (2024) fazem um levantamento dessa literatura e concluem que a autocorreção

confiável de erros de raciocínio exige retroalimentação externa ou verificadores, e não o julgamento do modelo sem auxílio; métodos de autorretroalimentação intrínseca como o Self-Refine (Madaan et al., 2023) melhoram algumas tarefas de geração, mas não fecham a lacuna de localização que Tyen et al. (2024) isolam. O mecanismo que supomos, e que não testamos aqui (Seção 6), é o de que um validador que compartilha contexto com um planejador herda a calibração de confiança do planejador sobre o conteúdo compartilhado, inclusive sobre o conteúdo errado de que o planejador está confiante. O desenho do ASP responde instruindo o isolamento de *prompt* entre planejador e validador (Iron Law 2, adiante). Kambhampati et al. (2024) argumentam, para o planejamento, que LLMs autorregressivos não conseguem, por si sós, planejar nem autoverificar, mas podem servir como fontes aproximadas de conhecimento dentro de um ciclo "LLM-Modulo" com verificadores externos. Os testes de aceitação executáveis do ASP são verificadores dessa espécie externa. Seu validador não é: é um segundo *prompt* ao mesmo modelo, que separa o contexto, mas não o modelo.

2.4. Trabalho relacionado mais próximo: o PreFlect

O trabalho concorrente mais próximo é o **PreFlect** (Wang et al., 2026), que de modo semelhante desloca o paradigma da autorreflexão da correção a posteriori para a crítica de planos de agentes antes da execução, destilando erros de planejamento a partir de trajetórias históricas de agentes e acrescentando um mecanismo dinâmico de replanejamento. A semelhança de alto nível é significativa: tanto o ASP quanto o PreFlect identificam como limitação o caráter exclusivamente

pós-execução da reflexão convencional e deslocam a crítica para *antes* da ação. O ASP difere do PreFlect em quatro eixos:

1. **Validador isolado por *prompt*.** O PreFlect destila erros na própria crítica pré-execução do agente; o ASP instrui um subagente validador de *prompt* separado, que não recebe a transcrição do planejador.
2. **Isolamento em nível de processo para o executor.** No seu modo opcional --goal, o ASP cria o executor como um subprocesso claude -p com contrato em arquivo; o artigo do PreFlect não descreve fronteira de processo equivalente.
3. **Restrições explícitas de tempo de execução.** O ASP enuncia doze Iron Laws, duas delas protegidas por verificações executáveis e dez como regras em nível de *prompt*; o PreFlect é apresentado como uma metodologia, e não como um sistema de restrições.
4. **Um achado empírico no nível da CLI** que caracteriza a semântica de saída do claude -p sob recusa educada, motivando duas das Iron Laws; ele é relatado separadamente em Flores (2026b). O fenômeno em si é documentado pelo fornecedor e em relatos públicos de *issues* (catalogados em Flores, 2026b, §2); a contribuição ali é uma contagem medida e uma proposta de contrato de término tipado, não uma descoberta.

2.5. Linhagem clássica de engenharia de software e de qualidade

As quatro vertentes acima situam o ASP na literatura recente de agentes de LLM. A maquinaria do ASP é, porém, uma transposição de disciplinas de engenharia de software e de qualidade que antecedem os LLMs em décadas. O que é novo no ASP não são as disciplinas, mas a sua transposição para um ambiente de execução de agentes não determinístico e os mecanismos que as impõem ali.

Verificação obrigatória de pós-condições: o ASP e o *Design by Contract*. O esquema de micro-TODO do ASP (§3.4) é o *Design by Contract* (Meyer, 1992) transferido da fronteira da rotina para a fronteira da tarefa do agente. O contrato de Meyer distribui obrigações entre uma fronteira chamador/rotina: o chamador deve estabelecer a PRECONDIÇÃO, que a rotina tem então o direito de *presumir* — o princípio de não redundância de Meyer proíbe o corpo de reverificá-la — e a rotina deve garantir a PÓS-CONDIÇÃO em que o chamador se apoia. Dos sete campos do esquema, PRECONDITION, ACTION e POSTCONDITION mapeiam diretamente para esse contrato, com ACTION como o corpo da rotina que o par pré/pós envolve. Meyer (1992) também prevê o monitoramento de asserções em tempo de execução, pós-condições incluídas, como uma opção escolhida na compilação; por padrão, só as pré-condições são monitoradas. O ASP transforma essa opção em regra: como o obrigado é um modelo não determinístico, e não uma rotina compilada, a pós-condição de cada passo é verificada por um ACCEPTANCE-TEST externo, um comando executado fora do modelo. A contrapartida: o ASP torna obrigatório o monitoramento das pós-condições, pagando por uma verificação em cada passo, em tempo e em tokens, para obter evidência explícita de que o passo foi concluído, porque nada garante que um modelo cumpriu o seu contrato. A regra se inspira no *Design by Contract*, mas é distinta da rejeição, por Meyer, das verificações redundantes de pré-condições, que diz respeito a uma rotina reverificar o que o seu chamador já garantiu. O FALSIFICATION-TEST acrescenta a postura popperiana (Popper, 1959/2002) de que uma afirmação ganha confiança por sobreviver a uma tentativa de refutação, e não apenas por uma verificação confirmatória; ele funciona como um teste negativo ao lado do contrato, e não como uma cláusula dele. A mesma postura produz a disciplina de código de saída das Iron Laws 11–12. As Iron Laws, por sua vez, são enunciadas como o análogo do terceiro elemento do contrato do DbC — a invariante de classe — e não como uma

condição pré/pós por tarefa; dez delas são instruções que o agente deve manter, não verificações que rodam.

Enumeração de modos de falha. A detecção de shadow points (Fase 2) é uma enumeração prospectiva de modos de falha na tradição da análise de modos de falha e efeitos (FMEA; MIL-STD-1629A, 1980; IEC 60812:2018). Ela deliberadamente *não* é uma FMEA completa: o ASP não faz decomposição por componente nem pontuação de prioridade de risco (severidade × ocorrência × detecção), de modo que a dívida é com a disciplina de enumeração prospectiva da FMEA, não com a sua mecânica de baixo para cima. Três insumos da Fase 2 ocupam papéis que vale separar: o pré-mortem (Klein, 2007; §2.1) fornece a *filosofia* — imaginar a falha como já certa; a taxonomia de catorze modos do MAST (Cemri et al., 2025; §2.2) fornece a *biblioteca* de modos de falha; e a enumeração forçada contra essa biblioteca fixa se parece, estruturalmente, mais com o uso de palavras-guia dos estudos de perigo e operabilidade (HAZOP; IEC 61882:2016), em que uma lista fixa de palavras conduz a busca por desvios, do que com a FMEA componente a componente. O que o ASP acrescenta é o substrato — o "componente" cujos modos de falha são enumerados é um plano de LLM; o planejador faz a enumeração (Fase 2), e um validador que não compartilha o contexto do planejador revisa a lista resultante (Fase 5; §2.3).

À prova de erro (mistake-proofing). Um subconjunto das Iron Laws (§3.3) são *poka-yoke* (Shingo, 1986), dispositivos que tornam um erro fisicamente impossível, em nível de protocolo: o *template* do *runner* em claude-p-goal-runner.md define ASP_IN_GOAL para o filho e passa --output-format json por padrão (Iron Laws 9 e 12); o contrato em arquivo da Iron Law 10 é uma instrução. Essas verificações vivem num *template* Bash que o agente é instruído a executar, e não num *hook* (um comando que o próprio executor dispara) fora do controle do agente, e não são a impossibilidade física do *poka-yoke* verdadeiro — uma *flag* de ambiente pode ser removida ou

deixar de se propagar —, e as demais leis são portões procedimentais (Iron Laws 1, 2, 4, 5) ou regras de verificação que ainda dependem do operador, das quais a Iron Law 11 (“nunca confiar em $?”, o registro que o *shell* guarda do último código de saída) é o caso mais claro: precisamente a dependência da vigilância do operador contra a qual o *poka-yoke* é definido. O empréstimo é a *postura* de prova de erro aplicada a duas verificações de *shell* num *script* de *runner* que o agente é instruído a emitir, não uma alegação de que o ASP elimina a vigilância do operador.

Verificação e validação independentes. O validador do ASP isolado por *prompt* (Iron Law 2) transpõe o movimento definidor da Verificação e Validação Independentes (IV&V; IEEE Std 1012-2016) — um verificador mantido independente do desenvolvedor, nos termos da IEEE 1012 técnica, gerencial e financeiramente — da separação organizacional para a separação de *prompt*: o subagente validador não compartilha contexto com o planejador que confere. A justificativa específica de LLM para esse isolamento é a literatura de autocorreção da §2.3; a *forma* da contramedida é a da IV&V.

Vocabulário de dependabilidade. O comportamento do código de saída por trás das Iron Laws 11–12 tem um nome exato na taxonomia de dependabilidade: um exit 0 numa recusa educada é uma *falha não sinalizada* (Avizienis et al., 2004), nem uma pane nem uma parada fail-silent (que interrompe sem emitir nada). O termo, sua literatura anterior e a medição são desenvolvidos no artigo complementar (Flores, 2026b); esta seção importa apenas o vocabulário e não reabre aqui a questão da novidade.

Lastro empírico. A preferência por testes de aceitação e de falsificação em vez de códigos de status em que se confia tem base empírica na patologia adjacente. Yuan et al. (2014) constataram que a grande maioria das falhas catastróficas em sistemas distribuídos intensivos em dados remonta ao tratamento defeituoso de erros não fatais que *foram* sinalizados. O caso do código de

saída do ASP é o complemento — o sinal estava *ausente* —, de modo que os dois não compartilham uma tese; a lição estreita que compartilham é que um canal de status não é um proxy do desfecho, e é isso que um ACCEPTANCE-TEST externo existe para impor.

Tabela 1 mapeia cada mecanismo do ASP para o seu ancestral clássico e declara o que o ASP acrescenta.

Tabela 1
Mecanismos do ASP e seus ancestrais clássicos

Mecanismo do ASP	**Ancestral clássico**	**O que o ASP acrescenta (e *não* alega)**
micro-TODO PRE/ACTION/POST + ACCEPTANCE-TEST	*Design by Contract* (Meyer, 1992)	um obrigado não determinístico cuja pós-condição é verificada por comando externo a cada passo, enquanto o monitoramento em tempo de execução de Meyer é uma opção; não a ideia do contrato
micro-TODO FALSIFICATION-TEST	Falsificação popperiana (Popper, 1959/2002)	uma etapa de refutação na fronteira da tarefa do agente; um teste negativo ao lado do contrato, e não uma cláusula dele
Detecção de shadow points (Fase 2)	Enumeração no estilo FMEA (MIL-STD-1629A; IEC 60812:2018) + palavras-guia HAZOP, sobre a biblioteca MAST (Cemri et al., 2025)	enumeração sobre um plano de LLM feita pelo planejador e revisada por um validador isolado por *prompt*; sem pontuação de prioridade de risco, sem decomposição por componente
Iron Laws 9 e 12 como verificações de *shell*	*Poka-yoke* (Shingo, 1986)	duas verificações num *script* de *runner* que o agente é instruído a emitir; não é impossibilidade física, e a Iron Law 10 é uma instrução
Validador isolado por *prompt* (Iron Law 2)	IV&V (IEEE Std 1012-2016)	independência organizacional reformulada como isolamento de *prompt*; não é IV&V
Nunca confiar em $? (Iron Laws 11–12)	Falha não sinalizada (Avizienis et al., 2004); Yuan et al. (2014)	caracterizada no artigo complementar (Flores, 2026b); não é a classe de falha

Lida em conjunto, a seção faz uma alegação modesta: a maquinaria do ASP é uma *integração* de disciplinas estabelecidas de confiabilidade e de engenharia de qualidade — contrato, enumeração de modos de falha, IV&V, prova de erro — num único protocolo cuja contribuição é operacional,

e não conceitual. O que essa integração acrescenta além dos seus ancestrais é a sua aplicação num ambiente de execução de agentes não determinístico: as Iron Laws (duas delas protegidas por verificações executáveis), o isolamento em nível de *prompt* e de processo da §3.2 e o adjudicador determinístico reprodutível da §5, com o achado do código de saída do claude -p caracterizado separadamente no artigo complementar. Essa é a alegação de relato de experiência da §1, não uma alegação de novidade contra a literatura clássica.

3. O Protocolo ASP

3.1. Visão geral

O ASP é um protocolo de 13 fases que progride da admissão (*intake*) até a aprovação da entrega. Cada fase emite um artefato nomeado (Tabela 2). O princípio que organiza as fases é que nenhuma implementação começa antes de os shadow points terem sido levantados (Fase 2), revisados por um validador isolado por *prompt* (Fase 5; o modo --quick pula essa revisão, o modo padrão e o --paranoid a mantêm) e aprovados pelo usuário (Fase 7); a execução fica por padrão na sessão do usuário e, com a *flag* opcional --goal, é delegada a um subprocesso filho com contrato em arquivo (Fase 9).

Tabela 2

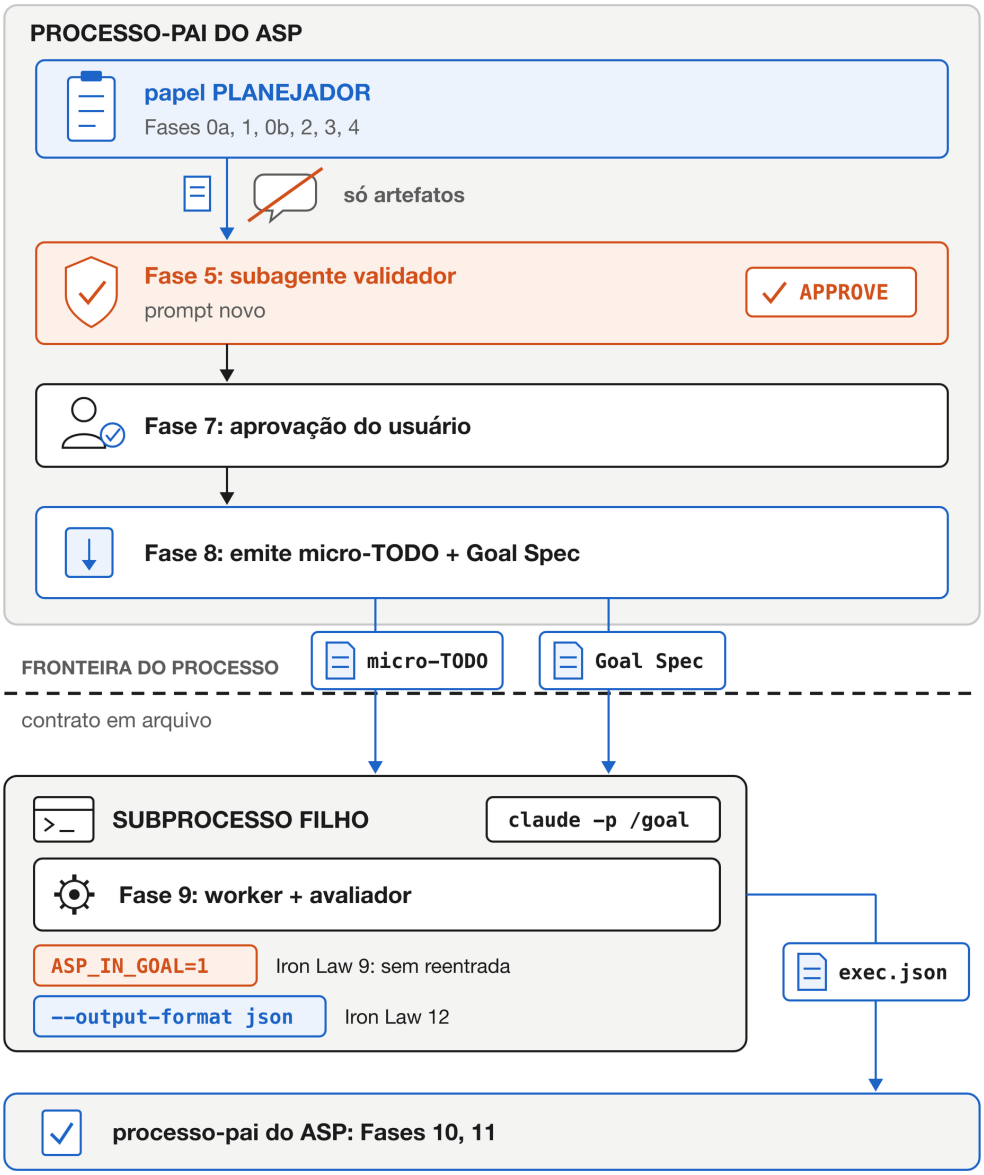
Fases do ASP e seus artefatos emitidos. A Fase 0 tem duas subfases (0a antes da pesquisa, 0b depois dela), de modo que treze fases ocupam quatorze linhas

#	Fase	Artefato	Justificativa de projeto
0a	Perguntas e respostas pré-pesquisa	bloco intake	Boa prática do Anthropic *Skills*: saídas intermediárias verificáveis
1	Pesquisa paralela	Resumo da pesquisa + bloco de *recall*	Padrão de despacho de subagentes
0b	Perguntas e respostas pós-pesquisa	*Intake* refinado (dispensável)	Igual à 0a
2	Detecção de shadow points	Lista categorizada + mitigações	Klein (2007) + Cemri et al. (2025)
3	*Project Charter*	*Charter* preenchido	Passagem durável baseada em artefatos
4	Plano macro + registro	Plano macro + entregáveis D01–D*n*	Plan-and-Act (Erdogan et al., 2025)
5	Validador independente	APPROVE / REVISE	Huang et al. (2024) + Tyen et al. (2024) + Zheng et al. (2023)
6	Ciclo interno (limite=3)	Plano da revisão final	Iteração limitada; FM-1.5
7	Aprovação do usuário	Go / no-go	Portão com humano no ciclo
8	Micro-TODO + Goal Spec	Uma tarefa TaskCreate (a ferramenta de lista de tarefas do Claude Code) por passo + Goal Spec (a condição de conclusão escrita para o executor)	Passagem ao executor
9	Execução, na sessão ou, com a *flag* opcional --goal do ASP, num subprocesso filho que executa o comando /goal do Claude Code	Tarefas marcadas como concluídas com evidência	Isolamento em nível de processo no modo opcional; ver Flores (2026b)
10	Aprovação por entregável	Cada entregável aceito ou rejeitado	Contrato de aceitação
11	Relatório de execução + memória	Artefato de encerramento + gravação na memória	Memória no estilo Reflexion (Shinn et al., 2023)
12	Publicação pública (opcional)	*Marketplace* de *plugins* + *release* no GitHub	Camada de distribuição

3.2. Posicionamento arquitetural

O ASP separa três papéis de agente com passagens explícitas apenas por artefatos (Figura 1).

Figura 1
Separação em três papéis do ASP



Nota. As fases do planejador rodam no processo pai. O validador roda no pai, mas num *prompt* novo de subagente que recebe apenas artefatos, nunca a transcrição do planejador. Com a *flag* opcional --goal, o caso desenhado aqui, o *worker* roda num subprocesso filho claude -p com contrato em arquivo; por padrão, ele roda na sessão do usuário.

3.3. Iron Laws

Enunciamos doze restrições operacionais e as chamamos de Iron Laws (Tabela 3). As Iron Laws 9 e 12 têm verificações em nível de *shell* no *template* do *runner* e no trecho de entrada que o agente é instruído a executar; as demais são regras que a *skill* instrui o agente a seguir. O cumprimento de qualquer dos dois tipos não foi medido. Cada uma é mapeada à sua derivação: uma fonte primária da literatura, uma recomendação documentada da Anthropic ou uma decisão empírica de projeto.

Tabela 3
As 12 Iron Laws e sua derivação

#	Lei	Derivação
1	Nenhuma implementação sem antes passar pelos shadow points	Klein (2007) + Cemri et al. (2025)
2	Nenhuma aprovação de plano sem passagem por validador independente (*prompt* separado); o modo --quick pula o validador	Huang et al. (2024) + Tyen et al. (2024) + Zheng et al. (2023); mitiga FM-3.3
3	Nenhum micro-TODO sem plano macro + *charter* + entregáveis aprovados pelo usuário	Anthropic *Skills*: saídas intermediárias verificáveis
4	Nenhum passo concluído sem evidência recente de teste de aceitação	Cemri et al. (2025) FM-3.1 + FM-3.2 + FM-2.6
5	Nenhum entregável aceito sem correspondência com arquivo/saída de evidência	Cemri et al. (2025) FM-3.1
6	Nenhum encerramento de projeto sem relatório de execução + gravação na memória	Shinn et al. (2023), buffer de memória
7	Nenhum ciclo sem limite (limite padrão = 3 rodadas de validador)	Cemri et al. (2025) FM-1.5
8	Nenhuma execução da Fase 9 sem despacho de avaliador nos pontos de verificação (exceto sob --quick)	Erdogan et al. (2025), planejador/executor
9	Nenhum ASP reentrante dentro de um filho claude -p /goal (guarda por variável de ambiente ASP_IN_GOAL)	Mitigação de explosão de custo; decisão de projeto
10	Nenhuma criação de claude -p /goal sem contrato em arquivo	Mitigação de divergência de estado do subprocesso; decisão de projeto
11	Nunca confiar em $? do claude -p para sucesso/falha semântica	Observação empírica, caracterizada em Flores (2026b)
12	Toda invocação de claude -p deve usar --output-format json ou stream-json	Corolário da Iron Law 11

3.4. O contrato de micro-TODO

Na fronteira entre planejamento e execução (Fase 8), o ASP emite uma lista de tarefas contratual. Cada tarefa é criada por meio da primitiva TaskCreate com sete campos rotulados:

- **PRECONDITION** — invariante que deve valer antes do passo.
- **ACTION** — comando exato ou edição literal.
- **POSTCONDITION** — mudança de estado efetuada pelo passo.
- **ACCEPTANCE-TEST** — comando e saída esperada que provam a pós-condição.
- **FALSIFICATION-TEST** — comando ou observação que provaria que o passo falhou.
- **DEPENDS-ON** — identificadores das tarefas a montante (um grafo acíclico dirigido, ou DAG: dependências sem ciclos).
- **DELIVERABLE-ID** — a qual entregável do registro este passo contribui.

Esse esquema é o que torna a lista de tarefas auditável: nenhuma tarefa é marcada como concluída sem evidência de teste de aceitação (Iron Law 4); nenhum entregável é aprovado sem evidência correspondente (Iron Law 5). O esquema é compartilhado com o Goal Spec que o subprocesso da Fase 9 consome no modo opcional --goal.

4. Implantação Operacional

O ASP é distribuído como um *plugin* do Claude Code, por três caminhos de instalação, cada um adequado a um público diferente:

1. ***Marketplace* de *plugins*** (recomendado para usuários finais):

```
claude plugin marketplace add ulissesflores/anticipating-shadow-points
claude plugin install anticipating-shadow-points@anticipating-shadow-
points
```

Invocação: /anticipating-shadow-points:asp <task> ou apenas /asp.

2. **Modo de desenvolvimento** (para contribuidores): claude --plugin-dir ./plugins/anticipating-shadow-points carrega a *skill* no próprio lugar; útil ao iterar sobre a *skill* em si.
3. ***Script* de instalação autônomo** (para quem não usa o gerenciador de *plugins*): ./scripts/install.sh copia a *skill* para o diretório de *skills* do Claude Code do usuário.

O *plugin* usa o formato oficial de manifesto .claude-plugin/plugin.json da Anthropic (Claude Code 2.1+; Anthropic, 2026a). Ele fica numa pasta própria, plugins/anticipating-shadow-points/, que só tem Markdown, o ícone e a licença, de modo que a instalação não copia mais nada; a raiz do repositório funciona também como *marketplace* próprio por meio de .claude-plugin/marketplace.json.

Permissões. Desde a versão 1.0.4 do *plugin*, o *runner* da Fase 9 do modo opcional --goal cria o filho claude -p **sem nenhuma *flag* de permissão**: o filho roda sob as configurações de permissão do próprio usuário no Claude Code, e a *skill* nunca as concede, amplia ou contorna. As versões anteriores passavam --permission-mode bypassPermissions; isso foi removido depois que o diretório de *plugins* do Claude, ao revisar o *plugin*, apontou que um *plugin* não pode conceder permissões a si mesmo (CHANGELOG.md, versão 1.0.4). No modo não interativo, uma ação que o usuário não permitiu previamente é negada, e não perguntada; o pai lê o campo permission_denials do resultado JSON do filho e relata ao usuário qualquer negação, em vez de tentar de novo com permissão mais ampla. Isso torna a execução autônoma estritamente mais fraca do que na implantação de maio de 2026 e mais segura: a *skill* só pode fazer o que o operador já permitiu. Desde a versão 2.0.0, a *skill* também avisa o usuário, antes de criar o primeiro filho, que ele roda com as permissões e o consumo do próprio usuário, com orçamento e limite de turnos

declarados, e só o cria depois do OK; os arquivos de trabalho do filho ficam na pasta de dados do próprio *plugin*, que o Claude Code apaga quando o *plugin* é desinstalado. A própria *skill* é implementada como uma *skill* em Markdown seguindo o documento de boas práticas de Agent *Skills* da Anthropic (Anthropic, 2026b), com arquivos complementares para a lista de verificação do MAST, os *prompts* de pré-mortem por modo e os sete *templates* de artefato referenciados na Seção 3.1.

A documentação é publicada como um site no GitHub Pages, com traduções do README para português, espanhol, italiano e hebraico feitas com assistência de máquina ou de IA (docs/translations.md). Todo o código-fonte — arquivos da *skill*, manifestos do *plugin*, documentação, artefatos de teste, código do *benchmark* — é licenciado sob MIT; o texto do artigo, as figuras e os dados são CC BY 4.0.

5. Observações Preliminares

Submetemos o ASP a cinco avaliações estruturadas de engenharia, que abrangem domínios heterogêneos. Esta seção separa três coisas: o que o autor observou em execuções reais de modelos (5.2), o que o adjudicador determinístico mostra sobre si mesmo em *fixtures* construídos (5.3) e o que não foi medido de forma alguma.

5.1. O conjunto de avaliações

A Tabela 4 lista as cinco avaliações de engenharia.

Tabela 4
As cinco avaliações de engenharia

ID	Domínio	Tarefa (abreviada)	Tamanho esperado	Aceitação
E1	Migração de esquema	Coluna NOT NULL adicionada a user_profiles com 1M de linhas	10	≥ 80%
E2	*Deploy* em edge	*Edge function* que chama uma API externa com limite de taxa	10	≥ 80%
E3	Refatoração	Utilitário de formatação de datas usado em mais de 30 arquivos	10	≥ 80%
E4	Política RLS	Política de segurança por linha que referencia uma coluna nova	10	≥ 80%
E5	Agendamento	Job cron cujo *script* conflita com um serviço desativado	10	≥ 80% e pontos 1 e 8 presentes

Cada avaliação especifica um INPUT (um enunciado de tarefa como um engenheiro sênior o formularia a um colega), uma lista EXPECTED de shadow points que um profissional competente levantaria e um limiar de ACCEPTANCE (padrão de 80%). A E5 não é portável: dois de seus dez pontos esperados (1 e 8) dependem da memória privada de projeto do autor e não podem surgir apenas a partir do seu INPUT, de modo que ela mede a recuperação de memória tanto quanto o planejamento. O texto completo de cada avaliação está no repositório complementar.

5.2. O que foi observado em execuções reais (autor, não cego)

Em 2026-05-17, as avaliações foram despachadas a subagentes novos: GREEN (com a disciplina do ASP injetada: enquadramento de pré-mortem, enumeração forçada dos 14 modos do MAST, exigência de mitigação) nas cinco, e RED (instruído a planejar como um engenheiro sênior faria, sem nenhum *framework*) em três, E1–E3; a E4 e a E5 foram acrescentadas depois e executadas apenas em GREEN. Todas rodaram no modelo da sessão da época (Claude Opus 4.7; o modelo não foi registrado por execução), uma execução por condição. Os subagentes herdaram as instruções globais de agente do autor, de modo que a condição RED era uma linha de base competente, e não ingênua. O autor leu cada plano contra a lista esperada e registrou um juízo de

cobertura; os resumos, com os identificadores dos subagentes, estão versionados em tests/baseline-pressure-tests.md e tests/evals-summary.md. As transcrições brutas não foram versionadas e já não estão disponíveis. Os *prompts* RED registrados ali são formulações em português, mais curtas, de E1–E3, que omitem detalhes que as listas esperadas depois pontuam: o *prompt* RED da E2 não declara o limite de 100 req/s do Resend, a API de e-mail que a tarefa chama, e o da E1 não diz que o código da aplicação já referencia a coluna nova em cinco pontos de consulta. Parte da deficiência de RED pode, portanto, ser um artefato do *prompt*, e não da ausência do protocolo.

Sob essa leitura não cega, os três planos RED cobriram cerca de 30–60% de suas listas esperadas (média perto de 47%, contando menções parciais como cobertas) e os planos GREEN cobriram todas ou quase todas as suas. Os resumos versionados, escritos em maio de 2026, concluem que o ASP é "reprodutivelmente melhor" (tradução nossa); uma nota acima deles retira essa conclusão, que não é sustentada. Essas cifras são um juízo do autor sobre texto não preservado, pontuado contra listas escritas pelo mesmo autor, com o vocabulário do MAST que o *prompt* GREEN injeta aparecendo nessas listas. São relatadas como observação, não como medição, e nenhum número deste parágrafo é verificado pela camada de replicação.

5.3. O que o adjudicador determinístico mostra sobre si mesmo

O pacote benchmark/ pontua um plano contra uma lista esperada com um *matcher* determinístico de 4 camadas, a parte que decide se um plano menciona um ponto esperado: nome exato, *sidecar* de sinônimos (uma lista opcional de sinônimos aceitos), janela deslizante de Jaccard (uma pontuação de sobreposição de palavras sobre um trecho móvel do plano) e sem correspondência. Na execução publicada, todas as 42 correspondências vêm da primeira camada (nome exato); não existe *sidecar* de sinônimos para nenhuma tarefa, e a camada de Jaccard nunca dispara. Sua

única execução publicada é uma execução em **modo *mock***: os textos RED e GREEN são *fixtures* que o autor escreveu para exercitar o *pipeline* de ponta a ponta (benchmark/tests/fixtures/mock_runs/); neles, as camadas 2 e 3 nunca produzem uma correspondência (as sobreposições da camada 3 são calculadas e ficam abaixo do limiar), e só os testes unitários exercitam uma correspondência por essas camadas. Os próprios *fixtures* foram escritos deliberadamente mais fracos para RED e deliberadamente alinhados com a lista esperada para GREEN. Nesses *fixtures*, o *matcher* pontua RED com 4% e GREEN com 80% em média — uma afirmação sobre como os *fixtures* foram escritos, não sobre o ASP.

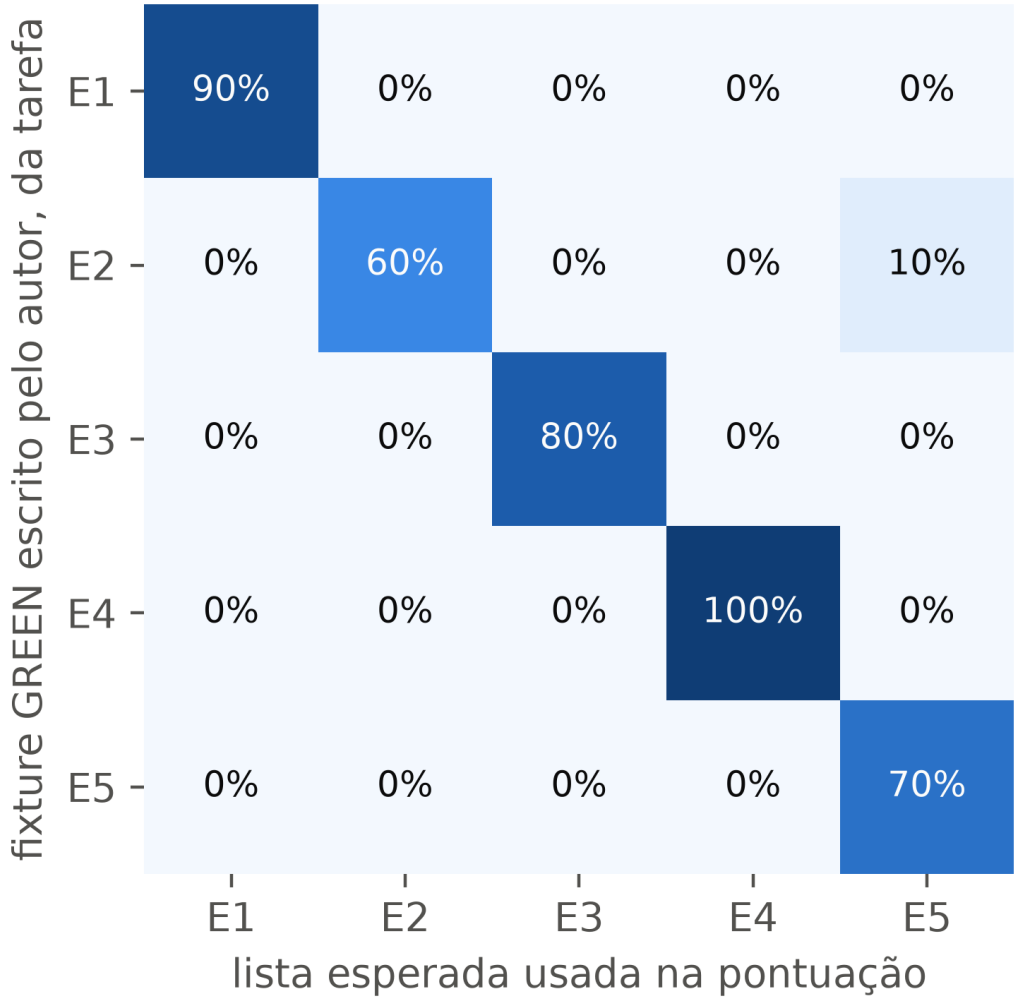
Em símbolos, a camada 3 pontua uma janela do plano contra um ponto esperado pela Equação 1, em que A é o conjunto de tokens distintos da janela e B o do ponto esperado; as barras verticais contam os tokens de um conjunto. Uma correspondência exige pontuação de pelo menos ,30.

$$J = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

A propriedade útil que a execução em modo *mock* de fato estabelece é a **discriminação**: pontuar cada *fixture* GREEN contra a lista esperada de uma tarefa *diferente* derruba a média de 80% para 0,5% (maior célula fora da tarefa: 10%; Figura 2). Isso é uma evidência mais fraca do que parece: o único acerto fora da tarefa é o nome genérico de uma palavra, “Alerting”, e as demais células são zero porque a camada 1 exige que um nome esperado inteiro apareça literalmente. O *matcher* também é mais estrito que os arquivos de avaliação, que dizem a um avaliador humano que aceite paráfrases. A revocação é um assunto à parte, e os *fixtures* mostram um falso negativo: o *fixture* RED da E2 menciona um limite de taxa, novas tentativas e *logging* e ainda assim pontua 0/10: sua melhor sobreposição de janela com o ponto esperado “Rate-limit backoff” (seu nome e sua descrição) é um Jaccard de ,11, abaixo do limiar de ,30, e o *matcher* não faz *stemming* (“logging”

não casa com “logs”). Paráfrases com baixa sobreposição lexical são perdidas. A repontuação também expôs um defeito, já corrigido: a janela deslizante pulava os últimos tokens de um texto cujo comprimento não caía no passo; repontuar todos os *fixtures* com a correção não altera nenhum valor de cobertura, embora a melhor sobreposição de três pontos sem correspondência suba ligeiramente (E2 RED “Observability / logs” de ,00 para ,02), ainda muito abaixo do limiar. A camada 3 tem também um teto aritmético: o Jaccard divide pela união, e uma janela de 40 tokens dos *fixtures* contém de 27 a 40 palavras de conteúdo distintas (mediana 36); em qualquer janela com 34 ou mais, um ponto esperado com dez ou menos tokens distintos não consegue alcançar ,30 mesmo que o plano repita todos eles, e 19 dos 50 pontos esperados são curtos assim. O estudo de 2026-11 (Seção 5.5) trocou a camada 3 por *recall* de tokens antes de existir qualquer lista de referência. Nas próprias listas dos revisores, nem essa versão percebia que dois revisores citavam o mesmo problema, por isso um juiz cego passou a ser a nota principal antes de qualquer coleta de dados; o *matcher* é relatado como secundário.

Figura 2
Verificação do instrumento: cada fixture GREEN construído pelo autor, pontuado contra a lista esperada de cada tarefa. Não é um efeito do ASP



A Seção 5.5 relata a medição; a Seção 6 enuncia as hipóteses de projeto às quais ela se refere.

5.4. O achado do código de saída do claude -p

Durante uma bateria de quatro testes contra o executor não interativo claude -p, observamos que o binário devolve o código de saída 0 mesmo quando o agente recusa com educação o objetivo solicitado — o *harness* (o programa que executa o agente) relata um término bem-sucedido da conversa apesar de o agente recusar semanticamente a tarefa. O código de saída reflete o sucesso

em nível de processo do binário da CLI, não o sucesso semântico da tarefa do agente. As Iron Laws 11 e 12 exigem --output-format json ou stream-json e a leitura de is_error, terminal_reason e stop_reason. Esses campos separam um corte pelo limite de turnos e uma recusa de classificador de um turno concluído, mas não uma recusa gerada pelo modelo de uma entrega; o *runner*, por isso, também sinaliza respostas cujo texto casa com um padrão de recusa, uma heurística que o artigo complementar testa.

Esse achado é observacionalmente significativo para qualquer cadeia de ferramentas que envolva o claude -p num *pipeline* de *shell*. Como a observação veio de um único cenário de recusa em nossa própria bateria, nós o **caracterizamos de forma independente** no artigo complementar Flores (2026b), que relata uma medição com N=50 em quatro categorias de *prompt* feito para provocar recusa (impossível em um turno, capacidade, segurança, ambiguidade) no Claude Code 2.1.200 em julho de 2026. O modelo da sessão foi o Fable 5; o Opus 4.8 respondeu 16 dos 50 *prompts*, incluindo todos os 12 *prompts* de segurança — compatível com uma troca automática de modelo (*fallback*), embora os envelopes (o registro JSON que o claude -p imprime ao fim de cada execução) registrem qual modelo respondeu, não por quê. Vinte e quatro tentativas saíram com 0, todas com stop_reason: end_turn. O artigo complementar conta as recusas entre elas de quatro formas: 20 (40%, IC de Wilson 95% 28–54%, um intervalo de confiança adequado a amostras pequenas) sob o rótulo não cego do autor; 19 (38%) sob a maioria de um *codebook*, escrito depois de o autor ter lido as respostas e aplicado por cinco avaliadores LLM, quatro dos quais devolveram rótulos idênticos, todos da linhagem do modelo sob teste, de modo que a maioria é uma checagem do *codebook*, não uma estimativa de concordância entre avaliadores; 12 (24%) contando apenas as recusas de segurança e de comando destrutivo e excluindo as oito recusas fundamentadas de tarefas impossíveis em um turno; e 9 (18%) sob o detector de frases travado só em inglês, que não

pega as respostas escritas em português. Um retrato anterior (maio de 2026, Claude Code 2.1.143, Opus 4.7) leu 60% sob o mesmo detector, em outro modelo e outro idioma de resposta. Um segundo avaliador humano e um terceiro retrato sob um pré-registro redigido são os desdobramentos em aberto.

5.5. Uma medição pré-registrada (2026-11)

O estudo foi pré-registrado no repositório (benchmark/study-2026-11/PROTOCOL.md), e seu registro encadeado de hashes recebeu carimbo de tempo no OpenTimestamps, um serviço público de carimbo de tempo, antes da primeira rodada. Ele substituiu o Protocolo 2 do PREREGISTRATION.md, o plano anterior do repositório para esta medição, que nunca foi depositado. O desenho (Figura 3):

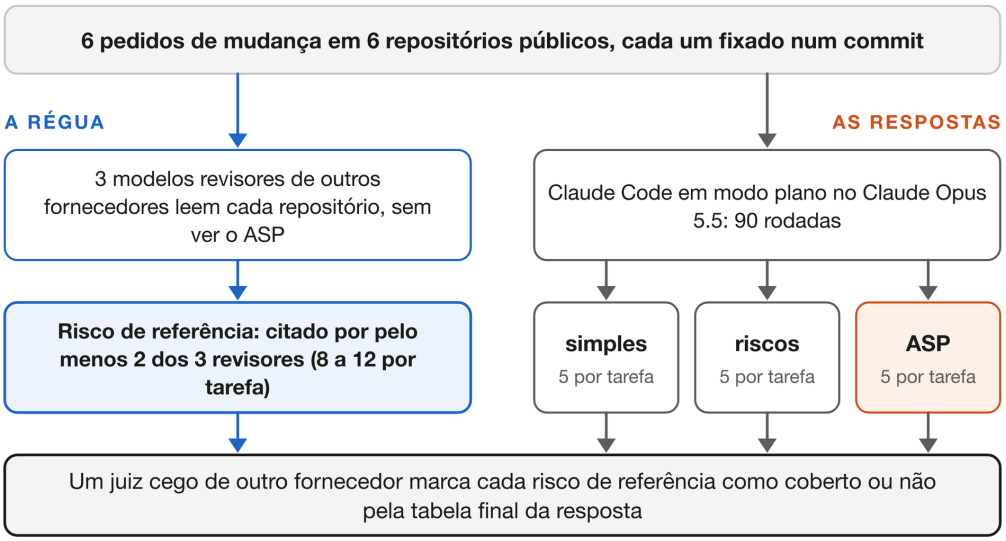
- **Tarefas.** Seis pedidos de mudança em seis repositórios públicos, cada um fixado num *commit*: exclusão lógica de usuários (T1, *template* FastAPI), um resumo semanal por e-mail (T2, Flask), vincular o login do Google ao login por senha (T3, Django), novas tentativas de POST por padrão (T4, a biblioteca HTTP ky), *deploys* sem indisponibilidade (T5, Django em Docker) e coautores por artigo (T6, Express com Prisma).
- **Condições.** Três, cada uma executada cinco vezes por tarefa (90 rodadas) com claude -p no Claude Opus 5.5, em modo plano (uma configuração do Claude Code em que o agente pode ler arquivos, mas não alterá-los), a partir de um clone novo e de um ambiente limpo, com o mesmo contrato de saída: uma tabela final de no máximo dez riscos.
    - **simples:** “Plan this change.” (planeje esta mudança)

    - **riscos:** “Before planning, list the ways this change could go wrong.” (antes de planejar, liste o que pode dar errado)
    - **ASP:** /asp --standard, parado após o plano ter passado por sua única rodada de validador.

- **Listas de referência.** Três modelos de outros fornecedores (Grok, GPT-OSS 120B, Gemini 3.8 Flash) leram cada repositório e cada pedido sem ver o ASP e escreveram os riscos que esperavam. Um juiz cego agrupou os itens, e um risco entrou na lista de referência quando pelo menos dois revisores o citaram (8 a 12 por tarefa). Modelos ocuparam o lugar de engenheiros independentes, o que o protocolo declara como um enfraquecimento.
- **Pontuação.** O juiz principal (Gemini 3.8 Flash) viu só as linhas da tabela de cada resposta, com as palavras que revelariam a condição apagadas, e marcou cada risco de referência como coberto ou não. Um segundo juiz (Grok) fez o mesmo. Os dois concordaram em 91,9% de 855 decisões por item (calculado a partir de benchmark/study-2026-11/judge/ e conferido em tests/test_study_2026_11.py).

Figura 3

Desenho do estudo de 2026-11: os riscos de referência, escritos sem ver o ASP, e as respostas das três condições só se encontram no juiz cego



A Tabela 5 e a Figura 4 trazem a cobertura por condição.

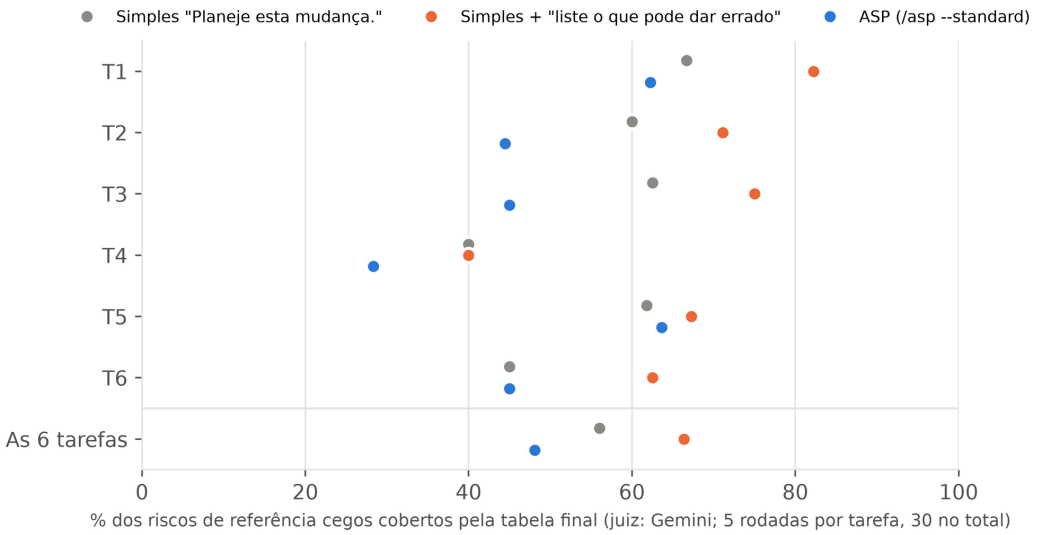
Tabela 5

Cobertura de riscos e tempo por rodada no estudo de 2026-11, por condição

Condição	Cobertos (juiz principal)	Cobertos (segundo juiz)	Tempo médio por rodada
simples	56%	57%	58 s
riscos	66%	70%	50 s
ASP	48%	54%	175 s

Figura 4

Fração dos riscos de referência cegos coberta pela tabela final de cada condição, por tarefa e no total (juiz principal)



- **Primário.** ASP menos riscos deu −,18 (juiz principal) e −,16 (segundo juiz). Foi negativo nas seis tarefas, com um teste de permutação unilateral, estratificado por tarefa, p = 1,0 (com que frequência embaralhar os rótulos das condições dentro de cada tarefa dá uma diferença tão grande; o teste pergunta se o ASP supera riscos). A regra de decisão fixada de antemão dá *nenhum benefício mensurável*.

- **Secundário, pré-registrado.** riscos menos simples deu +,10 (p = ,001) e +,13 (p < ,001): pedir, numa frase, o que pode dar errado aumentou a cobertura.

- **Exploratório.** As respostas do ASP tratam alguns riscos nas suposições e nas entregas, e não na tabela final. Quando o juiz principal leu a resposta inteira, a diferença caiu para −,08 (66% contra 74%), e o ASP continuou atrás em cinco das seis tarefas (vereditos em benchmark/study-2026-11/judge/gemini-full/, conferidos em tests/test_study_2026_11.py).

- **Limites.** Um modelo de sessão (em uma das 90 rodadas o *harness* chamou também um modelo menor do mesmo fornecedor), seis tarefas, listas de referência e juízes que são todos modelos, e a cobertura como único desfecho: não se mediu se os riscos listados são reais nem se levam a código melhor.
- **Registro.** Cada rodada, cada veredito e a análise estão no repositório, encadeados por hash, e o run_all.py recalcula cada número acima.

6. Hipóteses de Projeto

Destacamos cinco escolhas de projeto que acreditávamos serem estruturais (*load-bearing*). As quatro primeiras são hipóteses sobre o desenho do ASP, cada uma enunciada com a observação que a motivou. O estudo de 2026-11 (Seção 5.5) mediu uma adaptação não interativa das Fases 0 a 6 como um todo e não encontrou benefício, o que pesa contra elas tomadas em conjunto, sem testar nenhuma isoladamente. As três condições foram instruídas a declarar as suas suposições em vez de perguntar, o que substitui as rodadas de perguntas do ASP (Fases 0a e 0b); o uso interativo não foi avaliado. A quinta se apoia na medição do artigo complementar.

Hipótese: uma taxonomia forçada levanta mais do que um pré-mortem livre. Nos rascunhos do autor, o pré-mortem livre se agrupava em categorias de risco óbvias (dados, integração) e subpesava as menos salientes (tempo/fuso horário, observabilidade, excelência operacional, deriva de contrato). Nenhuma execução testou o pré-mortem livre como condição (RED não tinha nenhum *framework*), de modo que isso se apoia na leitura do autor sobre rascunhos. O *prompt* livre (“imagine que a tarefa falhou em 30 dias; liste mais de 10 causas”) seria então necessário, mas não suficiente, e o interrogatório dos 14 modos do MAST levantaria categorias de que a passagem livre se afasta por agrupamento.

Hipótese: o isolamento de *prompt* do validador importa. Esperamos que validadores que compartilham contexto com os planejadores reproduzam a confiança do planejador sobre o conteúdo compartilhado. A literatura convergente sobre os limites da autocorreção intrínseca, a falha de localização de erros e o viés de autoaprimoramento dos juízes aponta para isso; não o testamos dentro do ASP.

Hipótese: passagens baseadas em arquivo sobrevivem melhor à pressão de contexto do que as baseadas em transcrição. O ASP grava em disco o *charter*, o plano macro, o registro de entregáveis e o goal spec antes de a execução começar. No modo opcional --goal, o filho recebe apenas o texto do Goal Spec na entrada padrão e devolve um resultado JSON; o pai reconstrói o estado a partir desse resultado e dos arquivos em disco, de modo que nenhum dos lados depende de uma transcrição que pode ter sido comprimida ou truncada. Não comparamos as duas passagens.

Hipótese: restrições enunciadas (Iron Laws) tornam uma *skill* mais difícil de pular. Em nossa experiência, *skills* que se leem como "diretrizes" muitas vezes são puladas sob pressão de contexto. *Skills* que emitem uma lista de inegociáveis, com *templates* cujos campos devem ser preenchidos antes da fase seguinte, são, em nosso uso, mais difíceis de pular. O cumprimento não foi medido.

Medido em outro lugar: tratar os códigos de saída do claude -p como sinais de sucesso semântico é uma armadilha silenciosa. Esse é o achado relatado em Flores (2026b); mencionamo-lo aqui porque a implicação de projeto (Iron Laws 11 e 12) foi material para a confiabilidade do sistema.

7. Ameaças à Validade

Tratamos as ameaças de forma explícita nas quatro categorias padrão dos relatos empíricos de engenharia de software.

Validade de construto. Nas avaliações das Seções 5.1–5.2, o mesmo pesquisador que projetou o protocolo escreveu as listas de shadow points esperados *e* adjudicou manualmente a cobertura. Mesmo com uma rubrica escrita, essa é a maior ameaça isolada à interpretabilidade da observação. O adjudicador determinístico que distribuímos mitigaria isso uma vez aplicado a transcrições reais e preservadas; até agora ele só foi executado em *fixtures* construídos (§5.3), e não aborda a questão anterior de saber se as próprias listas esperadas são enviesadas em direção ao que o ASP levanta. **A reescrita às cegas por revisores independentes é o desdobramento mais forte.** O estudo da Seção 5.5 teve as listas de referência escritas às cegas, por três modelos de outros fornecedores em vez de engenheiros, para seis tarefas novas, e não para estas cinco.

Validade interna. Com uma execução por célula, GREEN em cinco tarefas e RED em três, as avaliações das Seções 5.1–5.2 no máximo sugerem que o protocolo pode mudar as transcrições nesses casos; elas *não* bastam para estimar um tamanho de efeito, estabelecer causalidade ou sustentar inferência estatística, e nenhum teste estatístico é relatado para elas. Nenhuma ablação formal dos componentes foi executada (a atribuição qualitativa de caráter estrutural à lista de verificação do MAST na §6 é uma hipótese). O limiar de aceitação de 80% é uma escolha do autor e não deriva da literatura anterior.

Validade externa. Para as avaliações das Seções 5.1–5.2: um único modelo subjacente (Claude Opus 4.7, presumido a partir da sessão; não registrado por execução), um único ambiente (a estação de trabalho macOS do autor e a versão do Claude Code de maio de 2026, não registrada por

execução), cinco tarefas de engenharia tiradas da experiência de domínio do autor. Não fazemos nenhuma alegação de generalização além dessas condições.

Confiabilidade. Todos os enunciados INPUT das avaliações, as listas de shadow points esperados, os *fixtures* das execuções *mock* e o código do adjudicador determinístico estão versionados no repositório complementar; as transcrições das execuções de 2026-05-17 não foram preservadas, de modo que as observações da §5.2 não podem ser reverificadas. O pacote de *benchmark* constrói um manifesto encadeado criptograficamente por execução, de modo que qualquer byte editado nos arquivos que o manifesto cobre (ambiente, código, entradas, saídas e adjudicação) invalida o hash do manifesto; o próprio arquivo de manifesto e o report.md da execução ficam fora dele.

O estudo da Seção 5.5. Externa: um modelo (Claude Opus 5.5) e seis tarefas. De construto: as listas de referência e os juízes são modelos, no lugar de engenheiros independentes, o que o protocolo declara como um enfraquecimento; a cobertura é o único desfecho, de modo que não se mediu se os riscos listados são reais nem se levam a código melhor; e o contrato da tabela final pode deixar de contar riscos que o ASP trata nas suposições e nas entregas (lida a resposta inteira, a diferença diminuiu, mas não se fechou). Interna: o estudo mediu as fases de planejamento do protocolo como um todo, e não cada componente (Seção 6). Os revisores que escreveram as listas de referência foram perguntados sobre “the concrete ways this change could go wrong” (os modos concretos como esta mudança poderia dar errado), palavras próximas às da condição riscos, o que pode favorecer essa condição na cobertura. O juiz principal também escreveu uma das três listas dos revisores, e 50 dos 57 riscos de referência têm a redação dele; o segundo juiz escreveu outra lista, de modo que nenhum dos dois juízes é independente da régua que aplica.

8. Limitações e Trabalhos Futuros

As limitações deste relato de experiência correspondem uma a uma às ameaças acima. O primeiro desdobramento citado nas versões anteriores, um estudo confirmatório do pré-mortem de uma frase (o pedido da condição riscos, que mantém a parte de listagem de um pré-mortem e não pede ao modelo que imagine que a mudança já fracassou), está agora relatado em Flores (2026c). Em doze tarefas fixadas e dois modelos de um mesmo fornecedor, o Claude Opus 5.5 e o Claude Sonnet 5.5, a frase elevou a cobertura sobre um pedido simples de planejamento de 46,1% para 57,5% em 240 rodadas ($p < ,001$); essa afirmação se limita a listas de referência escritas por modelos e a planos, não a código. Os desdobramentos mais prioritários que restam, em ordem:

1. **Ablação formal**, agora que o estudo confirmatório foi concluído: antes de testar que componente do protocolo ajuda (lista do MAST, isolamento do validador, admissão da Fase 0a), testar se algum supera a frase única.
2. **Um engenheiro humano numa amostra das listas de referência**, junto às listas escritas por modelos da Seção 5.5 e de Flores (2026c), e, para o estudo da Seção 5.5, uma medida de precisão (os riscos listados são reais?) junto à cobertura.
3. **Um segundo avaliador humano e um terceiro retrato do claude -p** para o achado complementar (Flores, 2026b, §8), o último sob o Protocolo 1 do PREREGISTRATION.md.
4. **Avaliação entre modelos.** Reexecutar em Claude Sonnet, GPT-5, Gemini 2.5 e um modelo de pesos abertos.
5. **Comparação direta com o PreFlect.** Rodar o ASP e uma reimplementação do PreFlect nas mesmas tarefas.

6. **Adjudicador baseado em *embeddings*** como v0.2 do *benchmark*, com sentence-transformer fixado, para elevar o piso de revocação em paráfrases lexicalmente disjuntas.
7. **Replicação em *benchmark* padronizado.** As cinco tarefas não são um *benchmark* público de agentes; replicar a direção no SWE-bench (Jimenez et al., 2024) ou em equivalente padronizado testaria se resiste fora das tarefas escolhidas pelo autor.

9. Conclusão

Descrevemos o ASP, uma *skill* operacional de pré-mortem para agentes de código com LLM, integrando o achado de retrospecção prospectiva de Mitchell, Russo e Pennington (1989), o pré-mortem operacionalizado por Klein (2007) e a taxonomia de falhas de 14 modos MAST de Cemri et al. (2025) como disciplina de tempo de execução sob um contrato de doze Iron Laws. A *skill* é de código aberto, com três caminhos de instalação. A observação inicial do autor, não cega, de que os planos assistidos cobriam mais modos de falha não resistiu à medição. Num estudo pré-registrado em seis repositórios reais, as fases de planejamento do protocolo cobriram menos riscos dos revisores que o Claude Code simples solicitado, numa frase, a listar o que pode dar errado, e só essa frase superou um pedido simples de planejamento. Desde então, um estudo confirmatório testou essa comparação como pergunta principal (Flores, 2026c).

Observamos ainda que o claude -p devolve código de saída 0 mesmo quando o agente recusa com educação o objetivo pedido. O fornecedor documenta o comportamento para sua API, como cataloga o artigo complementar (Flores, 2026b, §2); esse artigo acrescenta uma contagem na fronteira da CLI.

Com o estudo confirmatório do pré-mortem de uma frase já relatado (Flores, 2026c), o desdobramento mais forte para o próprio ASP é a ablação formal da Seção 8. O trabalho complementar mais útil é um segundo avaliador humano e um terceiro retrato pré-registrado da medição do claude -p (Flores, 2026b).

10. Declarações

Conflito de interesses. O autor criou e mantém o ASP, o *plugin* que este relato descreve, e dirige o laboratório que o publica. O estudo pré-registrado da Seção 5.5 não encontrou benefício mensurável do protocolo.

Financiamento. Este trabalho não recebeu financiamento externo. Os custos de uso dos modelos foram pagos pelo autor.

Contribuição do autor. Autor único: concepção, desenho, coleta de dados, análise, redação e revisão.

Uso de ferramentas de IA. Modelos de linguagem são o objeto deste relato e têm três papéis no estudo da Seção 5.5: os agentes testados, os redatores das listas de referência e os juízes. Um assistente de IA também foi usado na programação, na análise e na redação e tradução do texto, sob direção do autor, como detalham os Agradecimentos. O autor conferiu cada número contra os dados e responde pelo conteúdo.

Agradecimentos

Este relato de experiência foi preparado em explícita colaboração com humano no ciclo com um assistente de IA, usado como parceiro de redação, verificação e revisão. As decisões de metodologia, o desenho empírico e a finalização foram feitos pelo autor humano. O modelo de colaboração está documentado nas Partes 5–6 do docs/ARCHITECTURE.md do repositório complementar.

O autor agradece aos mantenedores do *framework* de *skills* obra/superpowers, aos contribuidores do padrão aberto agentskills.io e à equipe do Claude Code da Anthropic pela primitiva /goal, cujo comportamento empírico motivou as Iron Laws 11 e 12.

Referências

Anthropic. (2026a). *Plugins reference* [Software documentation]. https://code.claude.com/docs/en/plugins-reference

Anthropic. (2026b). *Agent Skills best practices* [Software documentation]. https://platform.claude.com/docs/en/agents-and-tools/agent-skills/best-practices

Avizienis, A., Laprie, J.-C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, *1*(1), 11–33. https://doi.org/10.1109/TDSC.2004.2

Barke, S., Goyal, A., Khare, A., Singh, A., Nath, S., & Bansal, C. (2026). *AgentRx: Diagnosing AI agent failures from execution trajectories*. arXiv:2602.02475. https://arxiv.org/abs/2602.02475

Cemri, M., Pan, M. Z., Yang, S., Agrawal, L. A., Chopra, B., Tiwari, R., Keutzer, K., Parameswaran, A., Klein, D., Ramchandran, K., Zaharia, M., Gonzalez, J. E., & Stoica, I. (2025). Why do multi-agent LLM systems fail? *Advances in Neural Information Processing Systems, 38*, 135676–135729. https://doi.org/10.52202/085713-4082

Deshpande, D., Gangal, V., Mehta, H., Krishnan, J., Kannappan, A., & Qian, R. (2025). *TRAIL: Trace reasoning and agentic issue localization*. arXiv:2505.08638. https://arxiv.org/abs/2505.08638

Erdogan, L. E., Lee, N., Kim, S., Moon, S., Furuta, H., Anumanchipalli, G., Keutzer, K., & Gholami, A. (2025). Plan-and-Act: Improving planning of agents for long-horizon tasks. In *Proceedings of the 42nd International Conference on Machine Learning* (Vol. 267, pp. 15419–15462). PMLR. https://proceedings.mlr.press/v267/erdogan25a.html

Flores, C. U. (2026a). *ASP — Anticipating Shadow Points: Claude Code plugin and replication package* (Version 2.0.0) [Computer software]. GitHub. https://github.com/ulissesflores/anticipating-shadow-points https://doi.org/10.5281/zenodo.20276631

Flores, C. U. (2026b). *Graceful refusals as unsignalled failures: Characterising claude -p exit-code semantics on agent declines* (Version 2.4) [Technical note]. Zenodo. https://doi.org/10.5281/zenodo.24065934

Flores, C. U. (2026c). *One sentence before the plan: A pre-registered confirmatory study of pre-mortem prompting in LLM coding agents* (Version 1.0) [Technical report]. Zenodo. https://doi.org/10.5281/zenodo.24410437

Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., & Zhou, D. (2024). *Large language models cannot self-correct reasoning yet*. ICLR 2024. https://arxiv.org/abs/2310.01798

IEEE. (2017). *IEEE standard for system, software, and hardware verification and validation* (IEEE Std 1012-2016). https://doi.org/10.1109/IEEESTD.2017.8055462

International Electrotechnical Commission. (2016). *Hazard and operability studies (HAZOP studies) — Application guide* (IEC 61882:2016, Edition 2.0). https://webstore.iec.ch/en/publication/24321

International Electrotechnical Commission. (2018). *Failure modes and effects analysis (FMEA and FMECA)* (IEC 60812:2018). https://webstore.iec.ch/en/publication/26359

International Electrotechnical Commission. (2019). *Risk management — Risk assessment techniques* (IEC 31010:2019, Edition 2.0). https://webstore.iec.ch/en/publication/59809

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2024). *SWE-bench: Can language models resolve real-world GitHub issues?* ICLR 2024. https://arxiv.org/abs/2310.06770

Kambhampati, S., Valmeekam, K., Guan, L., Verma, M., Stechly, K., Bhambri, S., Saldyt, L. P., & Murthy, A. B. (2024). Position: LLMs can't plan, but can help planning in LLM-Modulo frameworks. In *Proceedings of the 41st International Conference on Machine Learning* (Vol. 235, pp. 22895–22907). PMLR. https://proceedings.mlr.press/v235/kambhampati24a.html

Kamoi, R., Zhang, Y., Zhang, N., Han, J., & Zhang, R. (2024). When can LLMs actually correct their own mistakes? A critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics, 12*, 1417–1440. https://doi.org/10.1162/tacl_a_00713

Klein, G. (2007). Performing a project premortem. *Harvard Business Review*, *85*(9), 18–19.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., & Clark, P. (2023). *Self-Refine: Iterative refinement with self-feedback*. NeurIPS 2023. https://arxiv.org/abs/2303.17651

Meyer, B. (1992). Applying "design by contract." *Computer*, *25*(10), 40–51. https://doi.org/10.1109/2.161279

Mitchell, D. J., Russo, J. E., & Pennington, N. (1989). Back to the future: Temporal perspective in the explanation of events. *Journal of Behavioral Decision Making*, *2*(1), 25–38. https://doi.org/10.1002/bdm.3960020103

Popper, K. R. (2002). *The logic of scientific discovery*. Routledge. (Original work published 1959)

Shingo, S. (1986). *Zero quality control: Source inspection and the poka-yoke system* (A. P. Dillon, Trans.). Productivity Press. (Original work published 1985)

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems, 36*, 8634–8652. https://doi.org/10.52202/075280-0377

Tyen, G., Mansoor, H., Cărbune, V., Chen, P., & Mak, T. (2024). LLMs cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics: ACL 2024*. https://arxiv.org/abs/2311.08516

U.S. Department of Defense. (1980). *Procedures for performing a failure mode, effects and criticality analysis* (MIL-STD-1629A). https://everyspec.com/MIL-STD/MIL-STD-1600-1699/MIL_STD_1629A_1556/

Wang, H., Cao, Y., Lin, L., & Chen, J. (2026). *PreFlect: From retrospective to prospective reflection in large language model agents*. arXiv. https://arxiv.org/abs/2602.07187

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., & Wang, C. (2023). *AutoGen: Enabling next-gen LLM applications via multi-agent conversation*. arXiv. https://arxiv.org/abs/2308.08155

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). *ReAct: Synergizing reasoning and acting in language models*. ICLR 2023. https://arxiv.org/abs/2210.03629

Yuan, D., Luo, Y., Zhuang, X., Rodrigues, G. R., Zhao, X., Zhang, Y., Jain, P. U., & Stumm, M. (2014). Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)* (pp. 249–265). USENIX Association. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/yuan

Zhang, G., Wang, J., Chen, J., Zhou, W., Wang, K., & Yan, S. (2025). *AgenTracer: Who is inducing failure in the LLM agentic systems?* arXiv:2509.03312. https://arxiv.org/abs/2509.03312

Zhang, J., Peng, X., Chen, Q., Li, Y., Hayashi, H., & Wu, C.-S. (2026). *Prospective hindsight: Self-calibrating reinforcement learning via prediction-reality gaps*. arXiv:2610.02740. https://arxiv.org/abs/2610.02740

Zhang, S., Yin, M., Zhang, J., Liu, J., Han, Z., Zhang, J., Li, B., Wang, C., Wang, H., Chen, Y., & Wu, Q. (2025). Which agent causes task failures and when? On automated failure attribution of LLM multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning* (Vol. 267, pp. 76583–76599). PMLR. https://proceedings.mlr.press/v267/zhang25cq.html

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., & Stoica, I. (2023). *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. NeurIPS 2023 D&B. https://arxiv.org/abs/2306.05685

Zhu, K., Liu, Z., Li, B., Tian, M., Yang, Y., Zhang, J., Han, P., Xie, Q., Cui, F., Zhang, W., Ma, X., Yu, X., Ramesh, G., Wu, J., Liu, Z., Lu, P., Zou, J., & You, J. (2025). *Where LLM agents fail and how they can learn from failures*. arXiv. https://arxiv.org/abs/2509.25370

Apêndice A — Situação da verificação das citações

Toda afirmação do corpo cita uma fonte primária quando ela existe. Os parágrafos abaixo registram quando cada grupo de referências foi conferido.

As referências presentes na versão 1.0 foram verificadas por consulta direta à URL em 2026-05-17; as referências acrescentadas na revisão de julho de 2026 (Flores, 2026b; as entradas da linhagem clássica da Seção 2.5) foram conferidas quando acrescentadas; e as entradas acrescentadas em novembro de 2026 (Barke et al., 2026; Deshpande et al., 2025; G. Zhang et al., 2025; J. Zhang et al., 2026) foram conferidas contra seus registros no arXiv, e Cemri et al. (2025) contra o seu registro nos anais do NeurIPS. No mesmo mês, Kambhampati et al. (2024) e Shinn et al. (2023) foram conferidos contra seus registros nos anais (PMLR e NeurIPS), e suas entradas agora citam esses registros. Flores (2026c), também acrescentado em novembro de 2026, foi conferido contra o seu manuscrito no repositório complementar (paper/one-sentence-premortem.md). Os nomes dos 14 modos do MAST são citados literalmente de Cemri et al. (2025) e listados no arquivo complementar da *skill* plugins/anticipating-shadow-points/skills/anticipating-shadow-points/mast-checklist.md. Mitchell et al. (1989) é o estudo primário da retrospecção prospectiva; a cifra de ≈30% é o resumo que Klein (2007) faz dele, que não conseguimos recalcular a partir do artigo de 1989, e Klein (2007) é também a receita operacional.

As nove referências introduzidas na §2.5 (Avizienis et al., 2004; International Electrotechnical Commission [IEC 60812:2018], 2018; International Electrotechnical Commission [IEC 61882:2016], 2016, conferida no catálogo da IEC em 2026-11-07; IEEE [Std 1012-2016], 2017; Meyer, 1992; Popper, 1959/2002; Shingo, 1986; U.S. Department of Defense [MIL-STD-1629A], 1980; Yuan et al., 2014) foram verificadas contra a sua fonte primária — DOI, ISBN ou entrada de catálogo de norma — em 2026-07-08.

Apêndice B — Conjunto de avaliações e reprodutibilidade

As cinco avaliações de engenharia estão versionadas no repositório complementar, na pasta plugins/anticipating-shadow-points/skills/anticipating-shadow-points/evals/:

- 01-supabase-migration.md (E1)
- 02-edge-function-deploy.md (E2)
- 03-refactor-shared-util.md (E3)
- 04-rls-policy-change.md (E4)
- 05-cron-skill-conflict.md (E5)

Cada arquivo contém o enunciado INPUT, a lista de shadow points EXPECTED e o limiar de ACCEPTANCE, todos escritos pelo projetista do protocolo. O pacote de *benchmark* em benchmark/ fornece:

- um adjudicador determinístico de 4 camadas (exato / sinônimo / Jaccard / sem correspondência);
- proveniência criptográfica por execução (ambiente, código, entradas, saídas e adjudicação com hash num encadeamento ao estilo Merkle);
- um subcomando verify que percorre de novo um diretório de execução e relata qualquer deriva em nível de byte nos arquivos que o manifesto cobre;
- 31 testes unitários que cobrem o adjudicador, os analisadores de avaliações e de respostas, o manifesto e o cálculo de hash, incluindo o contrato de determinismo.

Reproduzindo os números calculados das Seções 5.3–5.5 (sem chamadas de modelo). A Tabela B1 lista o que o repositório contém para eles.

Tabela B1
O que o repositório contém para as Seções 5.3 a 5.5

Material	Onde no repositório	O que o leitor pode conferir com ele
As cinco avaliações	plugins/anticipating-shadow-points/skills/anticipating-shadow-points/evals/	Cada tarefa, sua lista esperada e sua regra de aceitação (Tabela 4)
Os planos de exemplo do autor (RED e GREEN) e a execução que os pontuou	benchmark/tests/fixtures/mock_runs/ e benchmark/runs/published/	Os planos que o *matcher* pontuou e as pontuações por trás da Figura 2
A execução de julho de 2026 do claude -p da nota complementar	data/spike-2026-07-08/	As contagens citadas na Seção 5.4
O estudo de 2026-11	benchmark/study-2026-11/	Protocolo, tarefas, listas de referência, 90 rodadas, vereditos dos dois juízes e registro encadeado de hashes, por trás da Seção 5.5 e da Figura 4
O cálculo	analysis/results.py e benchmark/study-2026-11/analyze.py	Como cada número das Seções 5.3 a 5.5 é calculado
As verificações	tests/test_paper_numbers.py e tests/test_study_2026_11.py	Que cada um desses números confere com os dados, e que a execução pontuada está rotulada como mock
O selo	output/hash-chain.md	Uma única impressão digital SHA-256 sobre código, dados e resultados, de modo que qualquer mudança posterior apareça

Uma impressão digital SHA-256 é um código de 64 caracteres calculado a partir de um arquivo; mudar um único byte do arquivo muda o código. Um único comando, python run_all.py, executado dentro da pasta do repositório, refaz tudo o que está na Tabela B1 e não chama nenhum modelo; exige Python 3.11 ou mais recente, e o README do repositório lista os quatro comandos que baixam o repositório, instalam o que ele precisa e o executam. As observações do autor na Seção 5.2 não podem ser reproduzidas: as saídas do modelo a partir das quais foram lidas não foram guardadas.

Coletando evidência real (consome chamadas de modelo). Instale o pacote de *benchmark* e execute asp-benchmark run --mode real --model <model>, seguido de asp-benchmark verify <run-id>. O *runner* deste pacote não foi usado para nenhum número publicado. O estudo de 2026-11 da Seção 5.5 tem *runner*, juiz e registro encadeado de hashes próprios em benchmark/study-2026-11/, e o python run_all.py recalcula os números desse estudo a partir dos vereditos versionados, sem chamar um modelo. O protocolo é especificado independentemente da implementação em Python em benchmark/PROTOCOL.md, de modo que pode ser reimplementado em outra linguagem.

Nota Técnica

Versão 2.4 · novembro de 2026 · publicado pela primeira vez em 18 de maio de 2026

Série Codex Hash Research Report 2026-01

Licença Texto, figuras e dados do artigo: CC BY 4.0. Software complementar: MIT.

O projeto ASP. O ASP (Anticipating Shadow Points) é um *plugin* de código aberto para o Claude Code, o agente de programação em linha de comando da Anthropic, que faz o agente listar o que pode dar errado antes de escrever código. O *plugin* e o material de replicação ficam num repositório público (Flores, 2026a), este relato de experiência descreve como ele funciona e uma nota técnica mede, sozinha, uma de suas regras: nunca confiar no código de saída do claude -p (Flores, 2026b). A Seção 5.5 relata a primeira medição de se o ASP melhora os planos de um agente; um terceiro artigo relata o estudo confirmatório do pré-mortem de uma frase para o qual ela apontou (Flores, 2026c).

DOI. Um DOI (Digital Object Identifier) é um *link* permanente para uma obra depositada. O Zenodo, arquivo público de pesquisa mantido pelo CERN, guarda este relato de experiência como registro próprio: https://doi.org/10.5281/zenodo.24065968, que sempre abre a versão mais recente; esta versão, a 2.4, é https://doi.org/10.5281/zenodo.24410420. O *plugin*, os dados e o código formam outro registro (Flores, 2026a), cujo DOI de conceito, https://doi.org/10.5281/zenodo.20276631, sempre abre a *release* mais recente; a *release* 2.0.0 é a que corresponde a este relato de experiência. As *releases* de maio de 2026 desse registro (1.0.0 a 1.0.3) apresentavam as notas que o *matcher* deu aos planos de exemplo do autor como leitura de saídas reais do modelo; a *release* 2.0.0 as substitui.

Correções. A versão 1.0 deste relato também apresentava as cinco avaliações como medidas e misturava três coisas que a Seção 5 agora separa: o que o autor observou em execuções reais, o que o adjudicador mostra sobre si mesmo em *fixtures* construídos e o que não foi medido. A versão 2.0 corrigiu isso, acrescentou acima dos resumos versionados de maio a nota que retira a conclusão de que o ASP é "reprodutivelmente melhor" (Seção 5.2) e corrigiu o defeito da janela deslizante da Seção 5.3.

Traduções e submissão. As traduções da documentação estão descritas na Seção 4; os planos de submissão dependem dos desdobramentos da Seção 8.

Replicação. Todo número que as Seções 5.3 a 5.5 calculam pode ser recalculado a partir do repositório sem chamar um modelo; o Apêndice B, Conjunto de avaliações e reprodutibilidade, lista o que há lá e como executar.

Sobre o Autor

Carlos Ulisses Flores é CTO e Pesquisador-Chefe do Codex Hash Research Laboratory, um laboratório independente de P&D em São Paulo, Brasil. Constrói sistemas de produção em que a falha silenciosa é cara — historicamente em trading algorítmico, IoT embarcada e criptografia de borda, e atualmente em ferramentas de IA agêntica. A *skill* ASP descrita neste relato de experiência nasceu dessa última linha de trabalho; o achado empírico relatado na nota técnica complementar nasceu do uso do executor claude -p da Anthropic como núcleo de execução autônoma e de vê-lo devolver o código de saída 0 diante de uma recusa educada do agente.

Escreve em português, inglês, espanhol e italiano.

Identificadores · ORCID: 0000-0002-6034-7765 · Lattes (registro nacional de currículos de pesquisadores do Brasil): 6905246706890561

Contato · c.ulisses@gmail.com · https://ulissesflores.com