CONTEXT ENGINEERING PLATFORM · PLAYBOOK

Quatro camadas e um harness decidem se os agentes entregam ou vão para o cemitério.

Agentes raramente falham em produção porque o modelo é fraco. Falham porque a engenharia ao redor do modelo nunca foi construída. O Context Platform Stack nomeia as quatro camadas, cloud e infraestrutura, engenharia de plataforma, engenharia de contexto e engenharia de intenção, e o harness que as opera como um único loop de entrega, aterrissado no GitHub Copilot: agent mode, arquivos de instruções, MCP servers, o model picker e orçamentos em GitHub AI Credits.

AUTORA

Paula Silva

CARGO

Software Global Black Belt

CONTATO

paulasilva@microsoft.com

TEMPO DE LEITURA

~ 45 minutos, fim a fim

4	6	62%
CAMADAS DE ENGENHARIA	CAPÍTULOS	DEPLOYMENT COM DISCIPLINA DE CAMADA

95%

PILOTOS SEM VALOR MENSURÁVEL

CAPÍTULOS

Chapter 00

O Context Platform Stack

O cemitério de agentes: 95% dos pilotos não entregam valor mensurável. As três causas raiz, a assimetria de benchmarks, as quatro camadas interdependentes e como a pilha se mapeia no GitHub Copilot.

Introdução →

Chapter 01

Fundação de cloud e infraestrutura

Onde os agentes rodam e a que custo real: Kubernetes e blocos do Azure, o pod de quatro containers, a anatomia de custo 42/58, identidade SPIFFE, mediação MCP e o caminho para a operação autônoma.

Camada 01 →

Chapter 02

Engenharia de plataforma e governança

Golden paths, guardrails, safety nets e review gates: como times de plataforma governam a entrega agêntica, RBAC por agente, o registro de MCP servers e o reality check do code review.

Camada 02 →

Chapter 03

Engenharia de contexto

Arquivos de instruções como infraestrutura, as três camadas de memória, disciplina de retrieval via MCP e orçamentos de contexto medidos em GitHub AI Credits: cure o menor conjunto de tokens de alto sinal.

Camada 03 →

Chapter 04

Engenharia de intenção

CONSTITUTION.md, especificações EARS, planos de implementação e spec-driven development com gate de aprovação humana que corta 73% dos defeitos de segurança.

Camada 04 →

Chapter 05

Integração e harness engineering

As quatro camadas como um único loop de entrega, roteamento de modelos por fase do SDLC, o harness de guias, sensores e evals, e a sequência de adoção de 90 dias.

Harness →

DICAS DE LEITURA

> **ATALHOS DE TECLADO**
>
> Aperte `/` para buscar. Use `j` e `k` para mover entre capítulos. Aperte `t` para alternar tema. Aperte `g` para ir ao topo.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

00 INTRODUÇÃO

O Context Platform Stack: por que agentes morrem antes da produção.

Agentes raramente falham em produção porque o modelo é fraco. Falham porque a engenharia ao redor do modelo nunca foi construída. Este capítulo apresenta o Context Platform Stack: quatro camadas interdependentes, infraestrutura, plataforma, contexto e intenção, que separam os pilotos que chegam à produção dos que engrossam o cemitério de agentes.

O cemitério de agentes: adoção quase universal, valor de um dígito

O State of AI in Business 2025 do MIT encontrou que 95% dos pilotos de GenAI enterprise não entregam valor mensurável, enquanto 78% das organizações relatam usar IA em pelo menos uma função, segundo a McKinsey. O Gartner projeta que 40% das aplicações enterprise terão agentes de IA para tarefas específicas até 2026, contra menos de 5% em 2025. Adoção quase universal de um lado, valor entregue de um dígito do outro. O gap entre a confiança executiva e a realidade do deployment não é um problema de tecnologia. É um problema de disciplina de engenharia, e é previsível.

O funil por trás das estatísticas é notavelmente consistente. Em um orçamento enterprise típico de 50 milhões de dólares para iniciativas de IA, 42 provas de conceito são financiadas em um ano fiscal. Onze chegam a piloto interno. Três chegam à produção servindo tráfego real. Uma gera receita ou redução de custo mensurável: 2,4% do investimento. Nos mesmos engajamentos, organizações com abordagem estruturada nas quatro camadas alcançaram 62% de deployment, uma ordem de magnitude acima da norma documentada pelo MIT. A diferença não é o modelo nem o tooling. É disciplina de camada: cada camada construída deliberadamente, com dono explícito e medição contínua.

Por que o contexto é o fator limitante

Três causas raiz explicam 80% das falhas. Infraestrutura tratada como commodity: cargas agênticas são stateful, ruidosas e escolhem ferramentas em runtime, então Kubernetes genérico funciona até uns três agentes e quebra previsivelmente a partir de trinta. Contexto tratado como detalhe: a frase que precede a falha é que o agente vai aprender sozinho, quando na realidade contexto é um artefato engenheirado, com camadas de memória, orçamento de tokens e curadoria humana semanal. Intenção tratada como opcional: nenhuma arquitetura sobrevive à diretiva de fazer o melhor possível, porque agentes precisam de uma hierarquia explícita de trade-offs ou aprendem uma implícita, geralmente a errada.

Um quarto fator amplifica os três: a assimetria dos benchmarks. Agentes demonstrados em tarefas de correção de bugs são 7 vezes mais fáceis de exibir do que agentes executando trabalho real de features. O mesmo modelo de fronteira que atinge 74,4% em correções de bugs no SWE-bench alcança apenas 11,0% no FeatureBench, em desenvolvimento de features complexas, e POCs construídos em torno de demos de bug fix superestimam sistematicamente a capacidade em produção. O contexto dá nome à pilha porque é onde os times menos investem: um agente sem contexto engenheirado é um chatbot com delírios de capacidade, e tudo o que o agente pode saber, quando pode saber e a que custo em tokens é decisão de projeto, não propriedade emergente.

O desenvolvimento AI-native também adiciona duas dívidas à dívida técnica familiar: a dívida cognitiva, quando desenvolvedores aceitam código gerado por IA sem entendê-lo, o estado que Storey (2026) chama de rendição cognitiva, e a dívida de intenção, quando objetivos, restrições e racional nunca são capturados e o agente otimiza a coisa errada assim que as condições mudam. Ignorar essas dívidas não adia o custo; distribui o custo para quem precisar manter, patrocinar ou depurar um agente que não sabe se explicar.

As quatro camadas, e como elas se relacionam

O Context Platform Stack define quatro camadas interdependentes, cada uma com dono, métrica e conversa de governança próprios. A Camada 01, Cloud e Infraestrutura, responde onde os agentes rodam e a que custo real: compute, GPU,

Kubernetes, padrões CNCF, observabilidade. A Camada 02, Engenharia de Plataforma, responde o que os agentes podem acessar e quem governa: o internal developer platform, golden paths, guardrails, RBAC por agente, quotas. A Camada 03, Engenharia de Contexto, responde o que os agentes podem saber, quando e a que custo em tokens: o padrão ACE, skills, três camadas de memória (hot, warm, cold), MCP servers. A Camada 04, Engenharia de Intenção, responde o que os agentes devem otimizar e em que hierarquia de trade-offs: CONSTITUTION.md, desenvolvimento orientado por especificação com sintaxe EARS, intent debt.

As camadas são codependências, não passos sequenciais. Cada camada serve a de cima e restringe a de baixo. Uma plataforma robusta sem captura de intenção produz caos em escala; intenção sem engenharia de contexto produz especificações que ninguém consegue implementar; contexto sem engenharia de plataforma produz ilhas de capacidade que não podem ser operadas. O corte em quatro camadas é deliberado nos dois sentidos: fundir contexto e intenção gera drift, porque qualquer mudança de skill passa a alterar o comportamento esperado e ninguém sabe se é bug ou feature, enquanto fragmentar infraestrutura e plataforma em cinco ou mais camadas dilui a responsabilidade sem ganho. O que está em jogo é concreto: um agente bem engenheirado fica em torno de 4.160 tokens por invocação, enquanto um mal engenheirado carrega tudo no contexto e paga 30.000 tokens por uma tarefa trivial.

> DEFINIÇÃO
>
> Quem controla o contexto do agente controla o seu comportamento; quem controla a sua intenção controla a sua estratégia; quem controla as suas especificações controla a sua escala (arXiv 2603.09619). Contexto, intenção e especificações não são refinamentos opcionais sobre a infraestrutura. Eles são a pilha, e a maturidade organizacional é o mínimo das quatro camadas, nunca a melhor delas.

O que este playbook cobre, pela lente do GitHub Copilot

Este playbook foi escrito para platform engineering leads e tech leads AI-native em ambientes enterprise, pessoas com 5 a 15 anos de experiência em sistemas, frustradas com o gap entre a promessa dos vendors e a realidade da produção. Os Capítulos 1 a 4 cobrem uma camada cada. O Capítulo 5 integra as camadas em uma única arquitetura. O Capítulo 6 cobre harness engineering, a disciplina de operação nomeada no início de 2026: como as quatro camadas compõem um harness de produção e como ele é governado ao longo do ciclo de vida do agente. Começando do zero, leia do 1 ao 4 em ordem, depois o 5, depois o 6. Já roda Kubernetes, comece pelo Capítulo 2 ou 3. Já opera agentes, comece pelo Capítulo 6.

A pilha não é abstrata, e este playbook a aterrissa na ferramenta que a maioria dos times já tem: o GitHub Copilot. O agent mode é a pilha em miniatura. Arquivos de custom instructions se comportam como hot memory, instruções por repositório e AGENTS.md curados são a curadoria de contexto warm, e MCP servers são a camada cold de retrieval com acesso governado. A escolha de modelo no picker do Copilot, entre Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro e a família GPT-5.x, é uma decisão de camada de plataforma com consequência direta de custo: desde 1º de junho de 2026, o Copilot cobra o consumo em GitHub AI Credits, em que um crédito corresponde a um centavo de dólar de uso, então uma camada de contexto sem gestão aparece na fatura em uma semana. Cada capítulo usa esses mecanismos do Copilot como exemplos recorrentes, ao lado de padrões neutros de vendor.

Cinco ações não exigem orçamento novo: catalogar os agentes que já estão em produção, medir o custo real de um agente por pelo menos uma semana, escrever o primeiro CONSTITUTION.md para o agente mais visível, nomear um dono para cada uma das quatro camadas e marcar uma revisão em 90 dias para medir o intent debt. O cemitério de agentes existe porque os times otimizaram para pilotos, não para produção. Este playbook inverte essa prioridade: leia, aplique e pare de construir agentes que você vai ter de enterrar.

Referências

Fontes primárias para o cemitério de agentes, o modelo de camadas e a engenharia de contexto.

- MIT NANDA, The GenAI Divide: State of AI in Business 2025, o estudo por trás dos 95% que enquadram o cemitério de agentes.
- Gartner, Press release sobre agentes de IA para tarefas específicas em aplicações enterprise, a projeção de 40% até 2026, contra menos de 5% em 2025.
- Context Engineering: From Prompts to Corporate Multi-Agent Architecture, arXiv 2026, a fonte do princípio de controle sobre contexto, intenção e especificações.
- FeatureBench: Benchmarking Agentic Coding for Complex Feature Development, arXiv 2026, a assimetria de benchmark entre correções de bugs e trabalho real de features.
- Anthropic, Effective context engineering for AI agents, orientação de prática sobre tratar contexto como artefato engenheirado.
- Model Context Protocol, o protocolo aberto por trás da camada cold de retrieval e do suporte a MCP no Copilot.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

01 CAMADA 01, CLOUD E INFRAESTRUTURA

A fundação de cloud e infraestrutura: onde os agentes rodam, e a que custo.

Antes de um agente poder saber qualquer coisa ou otimizar qualquer coisa, ele precisa rodar em algum lugar. A Camada 01 é o substrato: compute, rede, identidade e ambientes, dimensionados e governados para cargas agênticas, alimentando todas as camadas acima.

O gap de deployment: por que a camada de fundação existe

O State of AI in Business 2025 do MIT encontrou que 95% dos pilotos de GenAI enterprise não entregam valor mensurável, enquanto 78% das organizações relatam usar IA em pelo menos uma função. O funil por trás desse gap é previsível: em um orçamento enterprise típico de US$ 50 milhões para IA, 42 POCs financiados viram onze pilotos internos, três sistemas em produção e um que gera receita ou redução de custo mensurável, 2,4% do investimento. Organizações com uma abordagem estruturada para as quatro camadas do stack alcançaram uma taxa de deployment de 62%, uma ordem de magnitude acima dessa norma. A diferença não é o modelo nem o tooling. É disciplina de camada, e a disciplina começa por baixo.

A cadência de deployment conta a mesma história pelo lado das operações: cerca de 47% das organizações fazem deploy de modelos de IA apenas ocasionalmente e só 7% atingem deployment diário, a cadência que a melhoria contínua exige. O gap não é técnico. É infraestrutural: a maioria dos times tem compute, mas não tem a orquestração, a observabilidade, as fronteiras de segurança e a automação que transformam um artefato de modelo em um sistema de produção. A Camada 01 responde a uma pergunta: onde os agentes rodam, e a que custo. Para um time GitHub Copilot a pergunta é concreta. O agent mode no VS Code parece serverless, mas o Coding Agent executa em ambientes na nuvem, servidores MCP rodam como serviços reais, e tudo o que o Copilot constrói acaba aterrissando neste substrato.

O substrato: Kubernetes, blocos do Azure e extensões para agentes

Kubernetes é o runtime de fato para IA em produção: duas em cada três organizações rodando IA em escala o usam como plataforma primária de orquestração. No Azure isso normalmente significa AKS para o runtime dos agentes, Microsoft Entra ID com managed identities para identidade de workload, redes virtuais com private endpoints para que os agentes alcancem dados e ferramentas por caminhos privados, e ambientes separados de desenvolvimento, staging e produção para que um experimento nunca toque dados de produção. A consolidação está acelerando: em março de 2026 a CNCF quase dobrou o número de plataformas Kubernetes AI certificadas, mais de 70% de aumento nas certificações. Isso reflete dois fatos simultâneos: todos estão convergindo para a mesma plataforma, e o Kubernetes padrão precisa de extensões deliberadas para rodar cargas agênticas adequadamente.

As extensões existem porque agentes quebram as premissas da infraestrutura de ML tradicional. Um job de treinamento roda até terminar e um serviço de inferência responde requisições stateless; um agente roda por durações desconhecidas, ajusta recursos em tempo real, chama serviços externos com latência variável e executa código que não escreveu. O CNCF Kubernetes AI Conformance Program v1.35 nomeia três capacidades que tornam isso seguro: in-place pod resizing, para os recursos se ajustarem sem reinício quando um agente passa do planejamento para o uso intensivo de ferramentas; workload-aware scheduling, que quebra as esperas circulares que travam escalonadores FIFO quando agentes se coordenam; e sandboxing, isolamento rígido em nível de SO ou VM que resiste até a código de agente hostil. Os Cloud Native Agentic Standards de março de 2026 acrescentam a tríade operacional: least privilege, observabilidade MELT com métricas, eventos, logs e traces, e tolerância a falhas com retries e circuit breakers.

O pod de quatro containers

No nível do pod, uma carga agêntica de produção roda quatro containers, não um: o agent runtime com o loop do modelo e a lógica de decisão; um sidecar MCP que termina conexões MCP, aplica RBAC e rate limits de tokens e emite a trilha de auditoria; um sidecar de contexto servindo a camada de hot memory, o resolvedor de skills e o CONSTITUTION.md; e um sidecar de telemetria com o coletor

OpenTelemetry, decision spans e métricas MELT. Essa topologia é a primeira decisão de platform engineering que determina se o agente vai escalar.

Anatomia de custo, identidade e a fronteira de protocolo

A anatomia de custo de um agente em produção surpreende a maioria dos times. Nas linhas de base medidas em produção, 42% é compute, a parte que todos medem. Os 58% restantes ficam invisíveis até a fatura: 28% tokens de inferência, 15% memória e vector database, 10% rede e proxies MCP e 5% observabilidade, quase sempre subfinanciada. Um time que rastreia só compute tem uma estimativa de TCO errada por um fator de dois. O GitHub Copilot torna o vetor de tokens explícito: desde 1º de junho de 2026, o uso de modelos premium é cobrado em GitHub AI Credits, com 1 crédito = US$ 0,01, e cada sessão de agent mode ou tarefa do Coding Agent no picker, seja com Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro ou GPT-5.x, é uma linha medida. Trate esses créditos como gasto de infraestrutura, não como curiosidade por assento.

Identidade e mediação completam a camada. Cada agente carrega um SPIFFE SVID distinto, uma identidade criptográfica de workload, e cada permissão de recurso é uma policy em OPA: um agente sumarizador recebe acesso de leitura e quota diária de 50K tokens; um agente de incidentes SRE recebe acesso break-glass com approval gate em escritas. Misturar essas permissões em um único agente, o anti-padrão one-big-agent, significa que o primeiro incidente compromete tudo. O acesso em si flui pelo Model Context Protocol: agentes nunca tocam bancos ou APIs diretamente, falam MCP com resource servers que aplicam policy, registram acesso e limitam taxa na camada de protocolo. O MCP governa mais de 10.000 servidores ativos, com 97 milhões de downloads mensais de SDK no início de 2026, sob a Agentic AI Foundation na Linux Foundation, com mais de 170 organizações membras em março de 2026, e seu escopo é deliberadamente estreito: observabilidade pertence ao OpenTelemetry, identidade de workload ao SPIFFE/SPIRE, coordenação entre agentes ao A2A v1.0 e controle declarativo ao GitOps. Os servidores MCP que você configura hoje para o agent mode do Copilot falam o mesmo protocolo que os seus agentes de produção falarão.

DEFINIÇÃO

42% do custo de um agente em produção é compute. Os outros 58%, tokens de inferência, memória, rede e observabilidade, ficam invisíveis até a fatura chegar. Rastreie os cinco vetores ou a sua estimativa de TCO estará errada por um fator de dois.

De IaC à operação autônoma: como a camada alimenta o stack

Infraestrutura como código é o mínimo; o padrão que torna confiável a IaC gerada por LLM é o verificador no loop. O TerraFormer, da Georgia Tech com a AWS, roda terraform plan como feedback para o modelo e alcança melhoria de 15,94% no benchmark IaC-Eval, superando modelos 50 vezes maiores. O mesmo loop funciona no agent mode do Copilot: codifique as convenções de Terraform e de pipeline em .github/copilot-instructions.md, deixe o agente gerar, valide com o plan, itere. A supply chain é o risco dominante nesta camada: de 295 GitHub Security Advisories referenciando componentes LLM entre janeiro de 2025 e janeiro de 2026, supply chain responde por 44%, excessive agency por 20% e prompt injection por 18%. Nenhum scanner cobre todas as categorias, então combine pelo menos dois, como zizmor mais actionlint; com mediana de 2,71 segundos por workflow, não há desculpa de performance.

A trajetória da camada é a autonomia. Adotantes iniciais atingiram 95% de provisionamento automatizado em 2025, isso vira padrão entre empresas líderes em 2026, a gestão manual de infraestrutura se torna um passivo competitivo em 2027 e a operação totalmente autônoma é a linha de base enterprise em 2028. A arquitetura de referência para esse estado final é a Cognitive Platform Engineering, quatro planos sobre Kubernetes, Terraform e OPA, data, intelligence, control e experience, fechados por um loop Sense-Reason-Act: a plataforma sente padrões de workload, raciocina sobre a configuração ótima e age antes de alguém abrir um chamado. É assim que a Camada 01 alimenta o stack: a engenharia de plataforma herda primitivas de identidade e quota, a engenharia de contexto herda a mediação MCP e o substrato de memória, e a engenharia de intenção herda a trilha de auditoria que torna o drift mensurável. Infraestrutura não é encanamento, é política: cada decisão

de scheduling, caminho de rede e fronteira de isolamento aplica uma escolha sobre o que é permitido, visível e seguro.

Referências

Fontes do gap de deployment, dos requisitos do substrato e do stack de padrões.

- The GenAI Divide: State of AI in Business 2025, MIT NANDA, o dado de 95% de pilotos sem valor por trás do gap de deployment.
- CNCF Platforms Whitepaper, CNCF, a base de platform engineering sobre a qual as extensões agênticas se apoiam.
- Model Context Protocol, a fronteira de protocolo entre os agentes e a infraestrutura da qual dependem.
- Context Engineering: From Prompts to Corporate Multi-Agent Architecture, arXiv 2026, a arquitetura multiagente em camadas que esta camada de fundação sustenta.
- Effective context engineering for AI agents, Anthropic, a camada acima que este substrato serve.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

02 CAMADA 02, ENGENHARIA DE PLATAFORMA

Engenharia de plataforma: a camada de governança da entrega agêntica.

Agentes não falham em produção porque o modelo é fraco. Falham porque nada governa o que eles podem tocar. Golden paths, guardrails, políticas e review gates são a camada que transforma o agent mode do GitHub Copilot de risco operacional em capacidade de entrega governada.

A engenharia de plataforma virou a camada de governança

O State of AI in Business 2025 do MIT constatou que 95% dos pilotos corporativos de GenAI não entregam valor mensurável. Isso é um problema de governança, não de modelo, e a engenharia de plataforma foi onde a indústria buscou a resposta. A avaliação DORA de 2025 encontrou cerca de 90% das empresas já operando plataformas internas, e o Gartner projeta que 80% das grandes organizações de engenharia de software terão times de plataforma dedicados até 2026. No caminho, o papel mudou: times de plataforma deixaram de ser construtores de ferramentas internas e viraram arquitetos de governança.

O gatilho é a chegada dos agentes como operadores. Quando o agent mode do GitHub Copilot abre um pull request ou gera Terraform, ele toma decisões de infraestrutura com autonomia e exige a governança de um operador humano. Os resultados justificam o investimento: o time de plataforma do Spotify operou agentes que produziram mais de 1.500 pull requests aceitos, com 60% a 90% de economia de tempo em migrações, e a Thomson Reuters entregou ganhos de produtividade de 15x e 70% de automação em workflows críticos. O delta estrutural é pequeno: um IDP agêntico tem sete componentes, e apenas dois diferem de um IDP padrão, uma Agent API ao lado do portal humano e RBAC por agente ao lado das policies-as-code. Sem esses dois, a plataforma é boa para humanos e invisível para agentes.

DEFINIÇÃO

O princípio operacional deste capítulo, capturado na KubeCon EU 2026: agentes amplificam o que é bom no seu ecossistema e amplificam o que é ruim. Golden paths limpos e guardrails fortes são executados de forma impecável em escala; convenções inconsistentes e controles de custo fracos são replicados em milhares de deployments. Higiene de plataforma é pré-condição para agentes, não otimização.

Os quatro pilares: golden paths, guardrails, safety nets e review gates

O framework da CNCF identifica quatro pilares que times de plataforma precisam implementar para governar agentes autônomos: golden paths, guardrails, safety nets e workflows de revisão manual. No ecossistema GitHub Copilot, cada pilar se materializa em um mecanismo concreto: custom instructions, validação de políticas nos pull requests, resposta automatizada e review gates protegidos.

Golden paths: da intenção à infraestrutura

Golden paths definem o caminho abençoado para tarefas comuns: fazer deploy de um serviço, provisionar um banco, solicitar secrets. Caminhos escritos à mão não sobrevivem à velocidade dos agentes, então plataformas maduras os tornam generativos: o time de plataforma revisa e aprova o template do caminho uma vez, e os agentes o reutilizam milhares de vezes. Em termos de Copilot, o golden path mora onde o agente já olha, custom instructions do repositório que codificam os padrões do time e templates expostos via MCP, para que o agent mode consuma o blueprint aprovado em vez de improvisar um. O time de plataforma para de escrever Terraform o dia inteiro e passa a escrever schemas de validação.

Guardrails: do escaneamento reativo ao enforcement proativo

Guardrails impedem maus resultados antes da produção. O modelo histórico, scanners apontando violações depois do deploy, não funciona na escala dos agentes. Policy-as-code inverte a lógica: o time de plataforma descreve requisitos

de alto nível, conformidade com PCI-DSS, nenhum secret hardcoded, custo por serviço abaixo de US$ 10.000 por mês, e esses requisitos viram políticas determinísticas e executáveis que validam o código gerado por agentes antes da execução. Uma única definição de política cobre milhares de deployments conduzidos por agentes, e os revisores humanos focam em definir políticas e tratar exceções, não em validar caso a caso.

Safety nets: resposta autônoma

Safety nets capturam o que escapa dos guardrails. Quando um novo CVE aparece, a plataforma gera guardrails de runtime e os implanta em minutos, em vez das semanas que um patch coordenado em centenas de deployments costumava levar. A resposta autônoma a incidentes detecta anomalias, dispara troubleshooting conduzido por agentes e resolve incidentes sem intervenção humana, enquanto o SRE preditivo age antes que o usuário sinta o impacto.

Review gates: fricção estratégica

Nem tudo deve ser automatizado. Times de plataforma definem gates de revisão manual para decisões de alto risco: mudanças em sistemas de pagamento, migrações de banco de dados, expansões de quota de infraestrutura. Esses gates desaceleram os agentes de propósito. O auditor agent monta logs, diagramas de arquitetura e trilhas de auditoria em um pacote de compliance completo com score de risco, para que os revisores humanos gastem o tempo decidindo, não coletando informação.

Agentes como cidadãos de primeira classe da plataforma

Uma plataforma madura trata o agente como persona de usuário com identidade própria: cada agente carrega um SPIFFE SVID distinto, uma identidade criptográfica de workload, e cada permissão de recurso é uma política em OPA. Um agente de sumarização recebe acesso de leitura ao banco de produção e uma quota diária de tokens; um agente de incidentes de SRE recebe acesso break-glass emergencial com approval gate em operações de escrita. Misturar essas permissões em um único agente, o anti-padrão one-big-agent, significa que o primeiro incidente

compromete tudo. A mesma disciplina responde à tensão do vibe coding: um desenvolvedor pede ao agent mode do Copilot o deploy de um serviço, recebe código de infraestrutura de aparência plausível e o submete sem validar. A plataforma precisa assumir que todo artefato gerado por agente carrega risco: validação de política obrigatória antes do merge, análise de impacto de custo em cada deployment e rollback automático quando thresholds são violados.

O MCP é a interface que faz isso escalar. Em vez de construir APIs customizadas, o time de plataforma expõe capacidades como MCP servers, catálogo de serviços, provisionamento de infraestrutura, gestão de secrets, workflows de deployment, e mantém um registro descobrível deles. Quando um novo agente chega, ele descobre as capacidades pelo registro e opera dentro das restrições que encontra: o agente faz o próprio onboarding. Desde 1º de junho de 2026, o uso premium do Copilot é cobrado em GitHub AI Credits, um crédito equivale a US$ 0,01, consumidos na proporção dos tokens processados, o que torna orçamentos por agente aplicáveis em dólares. A execução rotineira de golden paths pode rodar em um modelo mais barato do picker, como o Haiku 4.5, enquanto Claude Fable 5 ou Opus 4.8 ficam reservados para o trabalho complexo que justifica o custo.

O reality check do code review

A narrativa da indústria sobre revisão de código autônoma não bate com os dados. Um estudo com 19.450 pull requests constatou que agentes de code review operando sozinhos alcançam 45,20% de taxa de merge contra 68,37% dos revisores humanos, e 60,2% dos PRs revisados apenas por agentes apresentam qualidade de sinal entre 0 e 30%. Uma análise de 278.790 conversas de revisão em 300 projetos explica por que os dois são complementares: comentários de IA são densos em defeitos, 29,6 tokens por linha contra 4,1 dos humanos, humanos acrescentam entendimento e transferência de conhecimento, e código de IA exige 11,8% mais rodadas de revisão. O pipeline segue direto daí: o Copilot Code Review faz a primeira passada, um humano faz a segunda, e nenhuma revisão de agente é jamais o único gate.

O merge também não é a linha de chegada. Uma análise de 1.210 PRs de correção de bugs gerados por agentes e aceitos, usando SonarQube, constatou que sucesso no merge não reflete qualidade pós-merge, com um agente acumulando em média

8,3 issues de análise estática por PR aceito. Um estudo com 33.000 PRs de cinco agentes mostrou que documentação e CI têm as maiores taxas de merge, performance e correção de bugs as menores, e que o modo de falha mais frequente é o abandono pelo revisor: PRs confusos, grandes demais ou mal descritos são simplesmente deixados de lado. A resposta da plataforma é específica: quality gate pós-merge obrigatório, limites de tamanho de PR e templates de descrição para código de agentes, e métricas de qualidade pós-merge no lugar da taxa de merge. Organizações que acertam essa camada relatam ganhos de produtividade de 2 a 5x nos times de engenharia.

> DEFINIÇÃO
>
> Antes de escalar agentes, seis elementos formam a plataforma mínima viável: um catálogo de serviços completo, golden paths desenhados para workloads de IA, um registro de MCP servers, RBAC por agente, observabilidade das trajetórias dos agentes que capture intenção e raciocínio, e governança de custo por agente em AI Credits. A falta de qualquer um dos seis é impedimento bloqueante para o deployment de agentes.

Referências

Fontes para a camada de engenharia de plataforma, os quatro pilares e a pesquisa de code review deste capítulo.

- CNCF Platforms Whitepaper, CNCF TAG App Delivery, a definição de referência de plataformas internas, golden paths e capacidades de plataforma.
- Model Context Protocol, o protocolo aberto pelo qual agentes descobrem as capacidades que um time de plataforma expõe, incluindo o agent mode do Copilot.
- The GenAI Divide: State of AI in Business 2025, MIT NANDA, o relatório por trás do número de 95 por cento de pilotos sem valor mensurável.
- From Industry Claims to Empirical Reality: Code Review Agents, Kennesaw State e Quanta, 19.450 pull requests, a lacuna de taxa de merge e qualidade de sinal entre agentes e humanos.

- Human-AI Synergy in Agentic Code Review, Queen's University, 278.790 conversas de revisão em 300 projetos, por que revisão de IA e humana são complementares.
- Beyond Bug Fixes: Code Quality After Agentic Merges, University of Saskatchewan, 1.210 PRs aceitos analisados com SonarQube, por que taxa de merge não é proxy de qualidade.
- Where Do AI Coding Agents Fail?, Drexel University, 33.000 PRs de cinco agentes, o abandono pelo revisor como modo de falha dominante.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

03 CAMADA 03

Engenharia de contexto: faça a curadoria do que o agente vê.

Arquivos de instruções, contexto do repositório, retrieval via MCP e orçamentos de tokens: a disciplina de dar ao agent mode do GitHub Copilot o menor conjunto de tokens de alto sinal que maximiza o resultado.

O que o agente vê é uma decisão de engenharia

Engenharia de contexto é a disciplina de estruturar o payload completo de informação que um agente recebe no momento da inferência. Onde o prompt engineering otimizava uma única interação, a engenharia de contexto pergunta o que o agente realmente precisa saber, em que ordem, a que custo, e como isso é gerenciado ao longo do tempo. Cada requisição que o agent mode do GitHub Copilot envia carrega arquivos de instruções, trechos do repositório, definições de tools, saídas de MCP e histórico da conversa, e desde 1º de junho de 2026 tudo isso é medido em GitHub AI Credits, com 1 crédito valendo US$ 0,01. A Anthropic formula o objetivo com precisão: encontrar o menor conjunto possível de tokens de alto sinal que maximize a probabilidade do resultado desejado.

Mais contexto não é contexto melhor. Pesquisadores analisaram 18 LLMs líderes e documentaram degradação severa e imprevisível à medida que o contexto cresce, o fenômeno conhecido como context rot. Em testes controlados, os mesmos 4K tokens de material de alta relevância superaram com frequência 100K tokens de material de relevância mista. A implicação é direta: curar o que o agente vê vale mais do que expandir o que ele poderia ver, e o ponto ótimo de compressão precisa ser medido por modelo e por workload, não presumido.

Quatro componentes que precisam manter coerência

A disciplina se apoia em quatro peças fortemente acopladas. Composição seleciona e formata o que entra: fatos, instruções, definições de tools, exemplos. Retrieval

fornece contexto em runtime via busca semântica e saídas dinâmicas de tools. Memória gerencia estado ao longo do tempo. Protocolos padronizam como agentes recebem e publicam contexto, principalmente o MCP. Em termos de Copilot, composição são os seus arquivos de instruções, retrieval é a busca no workspace mais os MCP servers, memória é o que persiste entre sessões do agente, e protocolos são a própria configuração de MCP.

Arquivos de instruções são infraestrutura, não prompts

O arquivo .github/copilot-instructions.md é carregado em cada requisição, o que transforma o tamanho dele em uma conta recorrente. Faça a matemática: um arquivo de instruções de 2.000 tokens, 1.000 desenvolvedores, 10 requisições por dia cada um. São 20 milhões de tokens por dia antes de uma única linha de trabalho real, cerca de US$ 60 por dia a US$ 0,003 por 1.000 tokens de input, ou 6.000 AI Credits. Arquivos .instructions.md com escopo por caminho via applyTo cortam esse custo em aproximadamente 68%: uma instrução sobre tratar falhas de pagamento carrega só para trabalho no subsistema de pagamentos. Combinado com skills de carregamento tardio, o mesmo dia cai para 6,4 milhões de tokens, cerca de US$ 19,20, e a diferença acumula perto de US$ 15 mil por ano nessa escala modesta.

A evidência mais forte é sobre quem escreve esses arquivos. Um experimento controlado em 47 repositórios internos mostrou que arquivos AGENTS.md curados por humanos reduziram o runtime em 28,64% e o consumo de tokens em 16,58%. Equivalentes gerados por LLM adicionaram 3,02% de overhead, pior do que não ter arquivo algum: especificações geradas automaticamente documentam o óbvio e omitem o crítico, o conhecimento tácito do time que só humanos identificam. A conclusão: arquivos de instruções são documentação de engenharia, com peer review, versionamento, tratamento de código de produção, owner nomeado e cadência de curadoria.

DEFINIÇÃO

Um arquivo de contexto tem exatamente dois estados: sinal curado ou ruído em acumulação. O teste é a propriedade. Se ninguém revisa mudanças no copilot-instructions.md em pull requests, o arquivo está derivando para o ruído, e cada desenvolvedor paga por essa deriva em AI Credits a cada requisição.

Camadas de memória e disciplina de retrieval via MCP

Um agente bem engenheirado carrega por camada, não por padrão. Memória hot, cerca de 660 tokens, está sempre no contexto: identidade do agente, convenções invioláveis, hooks de segurança, atualizada só em deploy. Memória warm, cerca de 2.000 tokens por skill ativada, carrega sob demanda: instruções especialistas, AGENTS.md por repositório, schemas de domínio, com curadoria semanal dos tech leads. Memória cold, cerca de 1.500 tokens por query, nunca é pré-carregada: knowledge base corporativa, dados de produção e código, recuperados via MCP servers com controle de acesso por agente. O resultado fica em torno de 4.160 tokens por invocação, enquanto um agente mal engenheirado despeja tudo na memória hot e paga 30.000 tokens por uma tarefa trivial.

O tiering não é teórico. Um deployment de produção aplicou essa arquitetura a um backend C# de 108.247 linhas ao longo de 12 semanas, cobrindo 283 sessões de agente e 2.801 prompts de usuário. Tokens por tarefa caíram 32%, de 18 mil para 12,2 mil por invocação, a latência p95 caiu 47%, de 9,2 para 4,9 segundos, e a conclusão de tarefas subiu 18%. Skills aplicam o mesmo princípio com lazy loading: descrições de 50 a 100 tokens ficam visíveis, o corpo completo só carrega na invocação; com cerca de 70% das skills instaladas sem uso em uma sessão, só isso corta o custo efetivo de contexto em 70%.

Contexto de MCP tem smells; audite

Cada MCP server habilitado no Copilot contribui descrições de tools para o contexto, e essas descrições sofrem do que pesquisadores chamam de tool description smells. Uma tool descrita como 'processa transações' convida o modelo a alucinar

os detalhes ausentes; uma spec sem documentação de erros falha em silêncio quando a tool real lança uma exceção. Auditar descrições em busca de vagueza, restrições de parâmetro ausentes e tratamento de erro faltante é engenharia de contexto pouco glamourosa e essencial. O ecossistema de skills mostra a negligência em escala: de 55.315 skills públicas pesquisadas, 26,4% não têm descrição de roteamento, o que as torna indescobríveis, e mais de 60% do conteúdo dos corpos é filler não acionável. Remover esse ruído comprimiu descrições em 48% e corpos em 39%, melhorando a qualidade funcional em 2,8%.

Orçamentos de contexto e higiene, medidos em AI Credits

Trate o orçamento de contexto como um orçamento de latência: um número explícito por invocação, acompanhado em AI Credits. Orçar, porém, não é encolher às cegas. O paradoxo da compressão está documentado: comprimir contexto agressivamente reduziu os tokens de input em 17% e aumentou o custo total da sessão em 67%, porque o modelo queimou muito mais tokens de output reconstruindo significado. A ideia governante é a densidade semântica. Tokens que carregam informação real, nomes descritivos, anotações de tipo, docstrings bem escritas, são investimentos; boilerplate e cerimônia são desperdício puro. Arquivos de instruções devem usar linguagem natural rica e clara em vez de abreviações, e cortes devem remover ruído, nunca sinal.

Higiene significa resistir a dois modos de falha ao longo do tempo. O brevity bias descarta insight específico de domínio em nome da compressão. O context collapse é pior: em uma execução documentada, um contexto de agente com 18.282 tokens atingia 66,7% de acurácia; reescrito por brevidade, caiu para 122 tokens e 57,1%, abaixo da linha de base sem contexto de 63,7%. A alternativa é curadoria deliberada: a abordagem ACE, geração, reflexão e curadoria sobre um contexto que evolui, entrega +10,6% de acurácia sobre linhas de base de contexto estático em benchmarks de agente e +8,6% em tarefas de análise financeira.

Combine profundidade de contexto com o modelo no picker

Combinar workload com modelo é engenharia de contexto, e o model picker do Copilot é onde isso acontece. Trabalho rápido de ação, edições e execuções de

tools, roda enxuto em Sonnet 5 ou Haiku 4.5: otimize latência, não profundidade de contexto. Raciocínio estendido sobre estado complexo, revisões de compliance, decisões de arquitetura, pertence a Claude Fable 5 ou Opus 4.8, onde 50K a 100K tokens por invocação podem ser gasto justificado. Compactação e handoffs para subagentes são território do Haiku 4.5: contexto mínimo, especificidade máxima. Parsing multimodal de PDFs, diagramas e tabelas pede um modelo com visão, como Gemini 3.1 Pro ou um GPT-5.x. Um workload de raciocínio em um modelo rápido entrega de menos; um workload de ação no Opus 4.8 superprovisiona AI Credits.

Referências

Fontes para a disciplina de curadoria, as camadas de memória e a economia de tokens.

- Effective Context Engineering for AI Agents, Anthropic, o enquadramento do menor conjunto de tokens de alto sinal.
- Agentic Context Engineering, arXiv, contextos que evoluem, brevity bias e context collapse.
- Codified Context: Infrastructure for AI Agents in a Complex Codebase, arXiv, os números de produção do three-tier em um backend C# de 108 mil linhas.
- A Survey of Context Engineering for Large Language Models, arXiv, o mapa de pesquisa de composição, retrieval, memória e protocolos.
- Model Context Protocol, MCP, o protocolo aberto por trás do retrieval de tools e dados no agent mode do Copilot.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

04 CAMADA 04, ENGENHARIA DE INTENÇÃO

Engenharia de intenção, expressar o que construir.

Contexto diz ao agente o que é verdade. Intenção diz o que ele deve querer. Este capítulo transforma propósito organizacional em artefatos que os agentes do GitHub Copilot conseguem ler, verificar e respeitar.

Contexto sem intenção otimiza a coisa errada

O caso Klarna, do terceiro trimestre de 2025, mostra o que acontece quando essa camada é pulada. O agente tinha tudo o que a engenharia de contexto pode oferecer, de dados detalhados de clientes a regras de preço e histórico de execução, e mesmo assim passou em semanas a otimizar métricas que a organização nunca quis priorizar; quando alguém percebeu, o desalinhamento já havia produzido custo operacional e reputacional substancial. Engenharia de contexto torna as decisões do agente mais precisas para um objetivo dado, mas não diz qual deve ser o objetivo. Um agente bem contextualizado sem intenção é um piloto com dados meteorológicos e especificações de combustível, mas sem plano de voo.

A pirâmide de quatro níveis

A hierarquia da fonte tem quatro níveis. Nível 1, prompt engineering: uma instrução para um modelo em uma interação; não escala entre sessões nem times. Nível 2, engenharia de contexto: cargas de informação, camadas de memória e retrieval via MCP, tema do capítulo anterior. Nível 3, engenharia de intenção: objetivos, valores e hierarquias de trade-off, respondendo o que o sistema existe para alcançar e quais restrições são inegociáveis. Nível 4, engenharia de especificação: um corpus legível por máquina de políticas, padrões de qualidade e restrições de implementação para operação multiagente coerente em escala. A maioria das organizações opera nos níveis 1 e 2; a distância até os níveis 3 e 4 é onde o intent debt se acumula.

Intent debt, uma dívida de clareza

Intent debt não é dívida de esforço, é dívida de clareza: acumula quando objetivos, restrições e o raciocínio por trás deles vivem em uma thread do Slack de oito meses atrás ou na cabeça de um único engenheiro, e não em artefatos duráveis. Três mecanismos a transformam em risco: concentração de conhecimento, quando saídas de pessoas deixam restrições sem documentação; drift de otimização, quando objetivos não declarados empurram humanos e agentes para proxies mensuráveis; e ausência composta, quando cada novo agente adiciona divergência sem nenhuma força puxando o sistema de volta. Quando agentes tomam decisões consequentes sem supervisão, essa dívida é risco operacional.

Três artefatos, e prompts como contratos

A engenharia de intenção se materializa em três artefatos de repositório. O CONSTITUTION.md é um documento de princípios e valores inegociáveis, escrito para humanos e estruturado para máquinas aplicarem. Cinco seções obrigatórias: Identity, com escopo e limites; Values and trade-offs, hierarquia ordenada no formato prefira X a Y, sempre; Hard constraints, o que o agente nunca faz; Protocols para situações recorrentes; e Escalation hooks, quando o agente para e chama um humano. Cerca de 200 linhas, com commit no Git.

SPECIFICATION.md em notação EARS

A especificação traduz princípios constitucionais em critérios de aceitação testáveis por máquina usando EARS, o Easy Approach to Requirements Syntax da ISO/IEC/IEEE 29148, criado para a Rolls-Royce em 2009. O requisito vago de resiliência a falhas de rede se torna: quando uma chamada de rede falhar 3 vezes em 60 segundos, o sistema deverá abrir o circuit breaker; enquanto o circuito estiver aberto, deverá servir respostas em cache por até 30 segundos. Requisitos deixam de ser prosa e viram condições que o agente verifica antes de executar.

IMPLEMENTATION_PLAN.md e o repositório Copilot

O plano quebra o trabalho em tarefas atômicas com marcadores [P] para o que pode rodar em paralelo e serve de fronteira de controle: o agente sabe exatamente o que pode modificar. No GitHub Copilot, o trio vive no repositório como contratos

versionados: custom instructions em .github/copilot-instructions.md levam os princípios constitucionais a cada requisição, prompt files reutilizáveis em .github/prompts/ codificam os rituais de especificação, e um agente customizado lê CONSTITUTION, SPECIFICATION e IMPLEMENTATION_PLAN antes de escrever código. Um prompt que importa não é texto descartável de chat; é um arquivo revisado, com dono e histórico de diffs.

> DEFINIÇÃO
>
> Um agente bem contextualizado sem intenção escrita é uma flecha com mira excelente e nenhum alvo. A engenharia de intenção fornece o mapa, o destino e os trade-offs que o agente está autorizado a fazer.

Spec-driven development, o fluxo executável

O Spec-Driven Development organiza o trabalho assistido por agentes em fases com gates de aprovação: humano e planner rascunham CONSTITUTION.md e SPECIFICATION.md no modo Ask, sem escrita de arquivos; a especificação é refinada em um IMPLEMENTATION_PLAN.md; segue o gate de aprovação humana; e só depois do sign-off explícito o agente entra no modo Agent, com escritas restritas ao plano e code review controlando o merge. O gate é fronteira técnica aplicada pela camada de orquestração, não convenção suave, e a pesquisa sobre IA constitucional reporta que o SDD reduz defeitos de segurança em 73% frente ao desenvolvimento assistido por IA sem restrições.

O gap planner-coder

Um estudo de falhas em sistemas multiagente encontrou que 75,3% se originam no gap entre planner e coder: especificações ambíguas ou incompletas viram código inseguro ou incorreto. O gap também é vetor de segurança: um atacante que influencia o planner via prompt injection pode fazer o coder gerar código vulnerável sem nunca tocá-lo. O SPECIFICATION.md é a interface entre planejamento e codificação; torná-lo explícito, interpretável por máquina e revisado por humanos fecha a superfície onde três quartos das falhas nascem.

Modos, modelos e AI Credits

O fluxo mapeia para o seletor de modelos do Copilot. Planeje no modo Ask com um modelo premium como o Claude Opus 4.8, ou o Claude Fable 5 para os trade-offs de arquitetura mais difíceis; implemente no modo Agent com o Claude Sonnet 5 como modelo de trabalho. Planejar faz poucas chamadas e custa poucos GitHub AI Credits, a unidade de cobrança desde 1º de junho de 2026, em que 1 crédito equivale a $0,01; executar faz muitas. Duas disciplinas protegem intenção e orçamento: hooks que bloqueiam escritas fora do plano rodam na camada de orquestração e consomem zero créditos, e o extended thinking fica no planejamento, porque em implementação iterativa ele adiciona 43% ao custo de tokens e degrada a qualidade em 30% em 4.018 trajetórias do SWE-Bench.

Verifique intenção, não sintaxe

Testes gerados por agentes não verificam intenção. Em um estudo com seis LLMs no SWE-bench Verified, 74,4% dos traces do agente mais forte incluíam testes escritos pelo próprio agente, mas eles funcionam como feedback observacional, mais próximos de print statements do que de assertions; o GPT-5.2 alcançou 71,8% de sucesso escrevendo quase nenhum teste. As preocupações precisam ficar separadas: o SPECIFICATION.md define o que deve ser testado, e um humano revisa se os testes gerados de fato testam isso. Sob mudanças semânticas, a taxa de aprovação de testes gerados por LLM cai de 79% para 66% e a cobertura de branches de 76% para 60%; depois de refatoração significativa, regenere e revise em vez de confiar neles.

As métricas de drift

Quatro sinais revelam drift cedo. A taxa de scope creep por sessão mede com que frequência o agente propõe trabalho fora do plano. O percentual de edições fora do IMPLEMENTATION_PLAN.md rastreia o que o enforcement dos hooks revela e deve tender a zero. O tempo de revisão humana por pull request gerado por agente é indicador antecedente de planos subespecificados. E os indicadores de dívida cognitiva, como tempo de onboarding, quantas pessoas conseguem explicar por que uma restrição existe e relutância em modificar certos arquivos, expõem intenção que nunca foi escrita.

Duas alavancas que os artefatos não substituem

Um estudo de 9.374 trajetórias em 19 agentes, 8 frameworks e 14 LLMs encontrou que o modelo determina o resultado mais do que o framework, e que trajetórias bem-sucedidas seguem a sequência entender, reproduzir, corrigir, verificar. A escolha no seletor do Copilot é decisão de governança de primeira ordem, e um hook que exige entendimento declarado antes de editar elimina uma classe inteira de falhas. A segunda alavanca é humana: IA durante o aprendizado reduz em 17% a aquisição de habilidades específicas de biblioteca, e os padrões que a preservam, investigação conceitual, código com explicação e geração seguida de compreensão, protegem a habilidade da qual a engenharia de intenção depende: escrever e avaliar especificações.

Referências

Fontes da pirâmide de quatro níveis, dos artefatos de intenção, do fluxo SDD e dos dados de verificação.

- Context Engineering: From Prompts to Corporate Multi-Agent Architecture, arXiv 2026, a hierarquia de quatro níveis de prompts a engenharia de especificação.
- Storey (2026), arXiv:2603.22106, intent debt, os três mecanismos de risco e as métricas de drift.
- Specification-Driven Development for AI-Assisted Software Engineering, SANER 2026, a estrutura de artefatos CONSTITUTION, SPECIFICATION e EARS.
- Constitutional AI reduces security defects in specification-driven development, a redução de 73% frente ao desenvolvimento assistido por IA sem restrições.
- Planner-Coder Gap in Multi-Agent Systems, os 75,3% das falhas multiagente nascidas entre planner e coder.
- Rethinking the Value of Agent-Generated Tests, seis LLMs no SWE-bench Verified, testes como feedback observacional.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

05 INTEGRAÇÃO E HARNESS

Integração e harness engineering: rode o stack como um único sistema.

As quatro camadas do Context Platform Stack só geram retorno quando rodam como um único loop de entrega, e esse loop só permanece confiável quando um harness de guias, sensores e evals envolve cada agente. Este capítulo monta o loop, nomeia o harness e fecha com uma sequência de adoção de 90 dias.

Um loop de entrega, quatro camadas

O Context Platform Stack não é um processo sequencial, é um grafo de dependências. Cloud e infraestrutura respondem onde os agentes rodam e a que custo. A engenharia de plataforma adiciona a governança, o internal developer platform, golden paths, RBAC e controles de custo que respondem o que os agentes podem fazer e quem pode fazer deploy. A engenharia de contexto adiciona a cognição, skills, camadas de memória e MCP servers que respondem o que o agente sabe. A engenharia de intenção adiciona a estratégia, CONSTITUTION.md e especificações que respondem o que o agente deve otimizar. O fluxo é bidirecional: a intenção restringe o contexto, o contexto determina quais capacidades de plataforma importam, e as escolhas de plataforma restringem a infraestrutura.
A maioria das falhas acontece nesses loops de feedback: infraestrutura construída sem intenção, contexto crescido sem governança, agentes lançados sem especificação.

Pular o loop leva ao cemitério de agentes, o repositório de protótipos abandonados que quase toda grande empresa acumula. O State of AI in Business 2025 do MIT encontrou que 95% dos pilotos de GenAI enterprise não entregam valor mensurável, enquanto organizações que trabalham as quatro camadas como um sistema alcançaram uma taxa de deployment de 62% nos engajamentos por trás deste playbook. Os benchmarks explicam parte do gap: o Claude Opus 4.5 atinge 74,4% no SWE-bench, correções de bug, mas apenas 11,0% no FeatureBench, o benchmark da ICLR 2026 com 200 tarefas em nível de feature, cerca de 7 vezes

mais difíceis que correções de bug. Um POC que só demonstra correções de bug opera na zona fácil e superestima a capacidade em produção.

> DEFINIÇÃO
>
> Cinco alertas precoces de que você está a caminho do cemitério: agentes sem SLOs nem acompanhamento diário de KPIs, nenhum owner nomeado pela evolução do agente, nenhum golden path para o próximo agente, contexto mantido por uma única pessoa e intent drift que passa despercebido por mais de duas semanas. Conserte a disciplina antes de escalar a frota.

Roteie modelos por fase do SDLC e pague em AI Credits

A escolha de modelo é específica por tarefa e por fase, não global, e o model picker do GitHub Copilot é onde a disciplina aterrissa. Especificação e arquitetura exigem decomposição sob incerteza, então roteiam para um modelo premium de raciocínio, Claude Opus 4.8 ou Claude Fable 5 com extended thinking, em modo ask. A implementação de features de 3 a 10 arquivos roteia para o Claude Sonnet 5 em agent mode com hooks habilitados. Docstrings, mensagens de commit e resumos de PR roteiam para o Claude Haiku 4.5. O Gemini 3.1 Pro e a família GPT-5.x cobrem os tiers equivalentes para times padronizados em outros fornecedores. A pesquisa por trás da divisão é específica: extended thinking custa 43% mais e degrada a qualidade em 30% em tarefas iterativas de implementação, então raciocínio premium dentro de loops apertados de código é ao mesmo tempo desperdício e pior.

Desde 1º de junho de 2026 essa tabela de roteamento também é um instrumento de orçamento, porque cada chamada premium no Copilot é cobrada em GitHub AI Credits e um crédito corresponde a um centavo de dólar de uso de modelo a preço de lista. Um time que reserva o Opus 4.8 para especificação e revisão, roda a implementação no Sonnet 5 e empurra resumos para o Haiku 4.5 paga por profundidade cognitiva exatamente onde ela compra qualidade. O mapa de maturidade diz por onde começar: o estágio 0 é o caos e o estágio 4 é intent driven, com todas as camadas ativas. Seu estágio é o mínimo das quatro camadas, não a

melhor delas: métricas DORA sem curadoria de AGENTS.md é estágio 1, e cinquenta agentes sem métricas de intenção ainda é estágio 2. Cada transição leva um ou dois trimestres.

O harness: guias, sensores e evals

A fórmula que cristalizou no início de 2026 é compacta: um agente é um modelo mais um harness. O modelo fornece o raciocínio; o harness é tudo ao redor, ferramentas, memória, estado, gates e validação. Mitchell Hashimoto cunhou engineer the harness em 5 de fevereiro. Em 11 de fevereiro, Ryan Lopopolo, da OpenAI, relatou três engenheiros entregando cerca de um milhão de linhas de código de produção com zero código escrito por humanos e nomeou a disciplina de harness engineering. Em 17 de fevereiro, a LangChain saltou do rank 30 para o rank 5 no Terminal Bench 2.0 mudando apenas o harness, com o modelo constante. Em 3 de abril, o Microsoft Agent Framework 1.0 chegou a GA com uma camada literalmente chamada Agent Harness. O modelo é o componente plugável; o harness é o diferencial.

Em março de 2026, Martin Fowler e Birgitta Böckeler organizaram o harness em guias, que direcionam o agente, e sensores, que o verificam. O padrão de três agentes da Anthropic mostra por que a separação importa: um Planner expande o prompt em uma especificação, um Generator implementa em sprints e um Evaluator navega pela aplicação como um usuário real antes de qualquer sprint fechar, com uma definição de pronto compartilhada e acordada de antemão. A pesquisa atribui 75,3% das falhas de agentes à interface planner coder, e esse padrão torna essa interface explícita e verificada.

Guias direcionam o agente

No GitHub Copilot os guias são arquivos concretos: .github/copilot-instructions.md como o mapa do repositório, arquivos .instructions.md com escopo por caminho, arquivos .prompt.md reutilizáveis para tarefas recorrentes e configurações de MCP servers que definem quais ferramentas existem. Mantenha o mapa em torno de 100 linhas apontando para docs mais profundas, um índice, não uma enciclopédia. E faça a curadoria à mão: em 47 repositórios internos, arquivos AGENTS.md curados

por humanos cortaram o runtime em 28,64% e o consumo de tokens em 16,58%, enquanto versões geradas por LLM adicionaram 3,02% de overhead.

Sensores verificam a saída

Sensores determinísticos são baratos e pegam a maioria das falhas: linters customizados que impõem direção de dependência, type checkers, testes estruturais, pre commit hooks, gates de CI que bloqueiam merges e detectores de drift de schema. Sensores inferenciais, revisão LLM as judge e a automação de code review do Copilot em pull requests, pegam o julgamento semântico que os linters não alcançam. A lição da LangChain: muitas vezes o problema não é o modelo, é o sensor que falta; detecção de loop e autoverificação obrigatória impulsionaram o salto deles no Terminal Bench. Evals fecham o conjunto: um golden set de prompts representativos roda no CI a cada mudança do harness e bloqueia regressões antes do ship.

A sequência de adoção de 90 dias

A definição de Hashimoto é a disciplina operacional: toda vez que o agente comete um erro, engenheire o ambiente para que esse erro se torne estruturalmente impossível de repetir. O loop tem seis passos: observe a falha em uma trajetória capturada, classifique a causa raiz, decida o tipo de correção, um novo sensor, ferramenta, atualização de instruções, gate ou subagente, implemente a correção, adicione um teste de regressão e audite as trajetórias futuras para confirmar que a correção segurou. Trajetórias capturadas desde o primeiro dia viram um ativo que compõe: cada sessão produz dados de treino, dados de eval e documentação de edge cases.

A sequência em si cabe em um trimestre. Dias 1 a 30: escolha um caso de uso com critério de sucesso verificável, escreva o .github/copilot-instructions.md, instrumente trajetórias, defina uma política mínima de aprovação e rode o primeiro experimento contra um baseline medido. Dias 31 a 60: plante sensores determinísticos, codifique as três a cinco regras arquiteturais mais importantes como mecanismos e adicione autoverificação mais revisão LLM as judge em pull requests. Dias 61 a 90: configure identidade e auditoria para os agentes em uso, replique o padrão em um segundo repositório e meça contra o baseline. Meça também além da

velocidade: um estudo com 2.989 desenvolvedores identificou seis dimensões que métricas no estilo DORA não capturam, autossuficiência, carga cognitiva, conclusão de tarefas, qualidade da revisão por pares, desenvolvimento de expertise e ownership do código.

Referências

Fontes primárias para o loop de integração e a disciplina de harness engineering.

- My AI Adoption Journey, Mitchell Hashimoto, 5 de fevereiro de 2026, o post que cunhou engineer the harness.
- Harness engineering: leveraging Codex in an agent-first world, Ryan Lopopolo, OpenAI Engineering Blog, 11 de fevereiro de 2026.
- Improving Deep Agents with harness engineering, LangChain, 17 de fevereiro de 2026, o estudo de caso do Terminal Bench 2.0.
- Harness engineering for coding agent users, Birgitta Böckeler, martinfowler.com, março de 2026, a taxonomia de guias e sensores.
- Harness design for long-running application development, Anthropic Engineering, março de 2026, o padrão Planner, Generator, Evaluator.
- Microsoft Agent Framework Version 1.0, Microsoft, abril de 2026, o release GA com a camada Agent Harness.
- FeatureBench: Benchmarking Agentic Coding for Complex Feature Development, arXiv 2026, o benchmark por trás do gap entre feature work e bug fix.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps