CONTEXT ENGINEERING PLATFORM · PLAYBOOK

Cuatro capas y un harness deciden si los agentes entregan o van al cementerio.

Los agentes rara vez fallan en producción porque el modelo sea débil. Fallan porque la ingeniería alrededor del modelo nunca se construyó. El Context Platform Stack nombra las cuatro capas, cloud e infraestructura, ingeniería de plataforma, ingeniería de contexto e ingeniería de intención, y el harness que las opera como un único loop de entrega, aterrizado en GitHub Copilot: agent mode, archivos de instrucciones, servidores MCP, el model picker y presupuestos en GitHub AI Credits.

AUTORA

Paula Silva

CARGO

Software Global Black Belt

CONTACTO

paulasilva@microsoft.com

TIEMPO DE LECTURA

~ 45 minutos, de punta a punta

4
CAPAS DE INGENIERÍA

6
CAPÍTULOS

62%
DEPLOYMENT CON DISCIPLINA DE CAPA

95%

PILOTOS SIN VALOR MEDIBLE

CAPÍTULOS

Chapter 00

El Context Platform Stack

El cementerio de agentes: el 95% de los pilotos no entrega valor medible. Las tres causas raíz, la asimetría de benchmarks, las cuatro capas interdependientes y cómo el stack se refleja en GitHub Copilot.

Introducción →

Chapter 01

Fundación de cloud e infraestructura

Dónde ejecutan los agentes y a qué costo real: Kubernetes y bloques de Azure, el pod de cuatro contenedores, la anatomía de costos 42/58, identidad SPIFFE, mediación MCP y el camino a la operación autónoma.

Capa 01 →

Chapter 02

Ingeniería de plataforma y gobernanza

Golden paths, guardrails, safety nets y review gates: cómo los equipos de plataforma gobiernan la entrega agéntica, RBAC por agente, el registro de servidores MCP y el reality check del code review.

Capa 02 →

Chapter 03

Ingeniería de contexto

Archivos de instrucciones como infraestructura, las tres capas de memoria, disciplina de retrieval vía MCP y presupuestos de contexto medidos en GitHub AI Credits: cure el menor conjunto de tokens de alta señal.

Capa 03 →

Chapter 04

Ingeniería de intención

CONSTITUTION.md, especificaciones EARS, planes de implementación y spec-driven development con gate de aprobación humana que recorta el 73% de los defectos de seguridad.

Capa 04 →

Chapter 05

Integración y harness engineering

Las cuatro capas como un único loop de entrega, ruteo de modelos por fase del SDLC, el harness de guías, sensores y evals, y la secuencia de adopción de 90 días.

Harness →

CONSEJOS DE LECTURA

> **ATAJOS DE TECLADO**
>
> Presiona `/` para buscar. Usa `j` y `k` para moverte entre capítulos.
> Presiona `t` para cambiar tema. Presiona `g` para ir al inicio.

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 INTRODUCCIÓN

El Context Platform Stack: por qué los agentes mueren antes de producción.

Los agentes rara vez fallan en producción porque el modelo sea débil. Fallan porque la ingeniería alrededor del modelo nunca se construyó. Este capítulo presenta el Context Platform Stack: cuatro capas interdependientes, infraestructura, plataforma, contexto e intención, que separan a los pilotos que llegan a producción de los que engrosan el cementerio de agentes.

El cementerio de agentes: adopción casi universal, valor de un dígito

El State of AI in Business 2025 del MIT encontró que el 95% de los pilotos de GenAI enterprise no entrega valor medible, mientras que el 78% de las organizaciones reporta usar IA en al menos una función, según McKinsey. Gartner proyecta que el 40% de las aplicaciones enterprise tendrá agentes de IA para tareas específicas en 2026, frente a menos del 5% en 2025. Adopción casi universal de un lado, valor entregado de un dígito del otro. La brecha entre la confianza ejecutiva y la realidad del deployment no es un problema de tecnología. Es un problema de disciplina de ingeniería, y es predecible.

El embudo detrás de las estadísticas es notablemente consistente. En un presupuesto enterprise típico de 50 millones de dólares para iniciativas de IA, se financian 42 pruebas de concepto en un año fiscal. Once llegan a piloto interno. Tres llegan a producción sirviendo tráfico real. Una genera ingresos o reducción de costos medible: el 2,4% de la inversión. En los mismos engagements, las organizaciones con un enfoque estructurado en las cuatro capas alcanzaron un 62% de deployment, un orden de magnitud por encima de la norma documentada por el MIT. La diferencia no es el modelo ni las herramientas. Es disciplina de capa: cada capa construida deliberadamente, con responsable explícito y medición continua.

Por qué el contexto es el factor limitante

Tres causas raíz explican el 80% de los fallos. Infraestructura tratada como commodity: las cargas agénticas son stateful, ruidosas y eligen herramientas en tiempo de ejecución; un Kubernetes genérico funciona hasta unos tres agentes y falla de forma predecible a partir de treinta. Contexto tratado como detalle: la frase que precede al fallo es que el agente aprenderá solo; en realidad, el contexto es un artefacto de ingeniería, con capas de memoria, presupuesto de tokens y curación humana semanal. Intención tratada como opcional: ninguna arquitectura sobrevive a la directiva de hacer lo mejor posible, porque los agentes necesitan una jerarquía explícita de trade-offs o aprenden una implícita, generalmente la incorrecta.

Un cuarto factor amplifica los tres: la asimetría de los benchmarks. Los agentes demostrados en tareas de corrección de bugs son 7 veces más fáciles de exhibir que los agentes que ejecutan trabajo real de features. El mismo modelo de frontera que alcanza el 74,4% en correcciones de bugs en SWE-bench llega apenas al 11,0% en FeatureBench, en desarrollo de features complejas, y los POCs construidos alrededor de demos de bug fix sobreestiman sistemáticamente la capacidad en producción. El contexto le da nombre al stack porque es donde los equipos menos invierten: un agente sin contexto de ingeniería es un chatbot con delirios de capacidad, y qué puede saber el agente, cuándo y a qué costo en tokens es una decisión de diseño, no una propiedad emergente.

El desarrollo AI-native también añade dos deudas a la deuda técnica conocida: la deuda cognitiva, cuando los desarrolladores aceptan código generado por IA sin entenderlo, el estado que Storey (2026) llama rendición cognitiva, y la deuda de intención, cuando objetivos, restricciones y lógica de decisión nunca se capturan y el agente optimiza lo incorrecto en cuanto las condiciones cambian. Ignorar estas deudas no aplaza el costo; lo distribuye a quien tenga que mantener, patrocinar o depurar un agente que no sabe explicarse.

Las cuatro capas, y cómo se relacionan

El Context Platform Stack define cuatro capas interdependientes, cada una con responsable, métrica y conversación de gobernanza propios. La Capa 01, Cloud e Infraestructura, responde dónde ejecutan los agentes y a qué costo real: compute,

GPU, Kubernetes, patrones CNCF, observabilidad. La Capa 02, Ingeniería de Plataforma, responde a qué pueden acceder los agentes y quién lo gobierna: el internal developer platform, golden paths, guardrails, RBAC por agente, cuotas. La Capa 03, Ingeniería de Contexto, responde qué pueden saber los agentes, cuándo y a qué costo en tokens: el patrón ACE, skills, tres capas de memoria (hot, warm, cold), servidores MCP. La Capa 04, Ingeniería de Intención, responde qué deben optimizar los agentes y en qué jerarquía de trade-offs: CONSTITUTION.md, desarrollo orientado por especificación con sintaxis EARS, intent debt.

Las capas son codependencias, no pasos secuenciales. Cada capa sirve a la superior y restringe a la inferior. Una plataforma robusta sin captura de intención produce caos a escala; la intención sin ingeniería de contexto produce especificaciones que nadie puede implementar; el contexto sin ingeniería de plataforma produce islas de capacidad que no se pueden operar. El corte en cuatro capas es deliberado en ambos sentidos: fusionar contexto e intención genera drift, porque cualquier cambio de skill pasa a alterar el comportamiento esperado y nadie sabe si es un bug o una feature, mientras que fragmentar infraestructura y plataforma en cinco o más capas diluye la responsabilidad sin beneficio. Lo que está en juego es concreto: un agente bien diseñado se mantiene en torno a 4.160 tokens por invocación, mientras que uno mal diseñado carga todo en el contexto y paga 30.000 tokens por una tarea trivial.

> DEFINICIÓN
>
> Quien controla el contexto del agente controla su comportamiento; quien controla su intención controla su estrategia; quien controla sus especificaciones controla su escala (arXiv 2603.09619). Contexto, intención y especificaciones no son refinamientos opcionales sobre la infraestructura. Son el stack, y la madurez organizacional es el mínimo de las cuatro capas, nunca la mejor de ellas.

Qué cubre este playbook, a través de la lente de GitHub Copilot

Este playbook está escrito para platform engineering leads y tech leads AI-native en entornos enterprise, personas con 5 a 15 años de experiencia en sistemas, frustradas por la brecha entre la promesa de los vendors y la realidad de producción. Los Capítulos 1 a 4 cubren una capa cada uno. El Capítulo 5 integra las capas en una sola arquitectura. El Capítulo 6 cubre harness engineering, la disciplina de operación nombrada a inicios de 2026: cómo las cuatro capas componen un harness de producción y cómo este se gobierna a lo largo del ciclo de vida del agente. Si empiezas desde cero, lee del 1 al 4 en orden, luego el 5 y el 6; si ya operas Kubernetes, empieza por el Capítulo 2 o 3; si ya operas agentes, ve directo al 6.

El stack no es abstracto, y este playbook lo aterriza en la herramienta que la mayoría de los equipos ya tiene: GitHub Copilot. El agent mode es el stack en miniatura. Los archivos de custom instructions se comportan como hot memory, las instrucciones por repositorio y los AGENTS.md curados son la curación de contexto warm, y los servidores MCP son la capa cold de retrieval con acceso gobernado. La elección de modelo en el picker de Copilot, entre Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro y la familia GPT-5.x, es una decisión de capa de plataforma con consecuencia directa de costo: desde el 1 de junio de 2026, Copilot factura el consumo en GitHub AI Credits, donde un crédito corresponde a un centavo de dólar de uso, así que una capa de contexto sin gestión aparece en la factura en una semana. Cada capítulo usa estos mecanismos de Copilot como ejemplos recurrentes, junto a patrones neutrales de vendor.

Cinco acciones no requieren presupuesto nuevo: inventariar los agentes que ya están en producción, medir el costo real de un agente durante al menos una semana, escribir el primer CONSTITUTION.md para el agente más visible, nombrar un responsable para cada una de las cuatro capas y programar una revisión en 90 días para medir el intent debt. El cementerio de agentes existe porque los equipos optimizaron para pilotos, no para producción. Este playbook invierte esa prioridad: léelo, aplícalo y deja de construir agentes que tendrás que enterrar.

Referencias

Fuentes primarias para el cementerio de agentes, el modelo de capas y la ingeniería de contexto.

- MIT NANDA, The GenAI Divide: State of AI in Business 2025, el estudio detrás del 95% que enmarca el cementerio de agentes.
- Gartner, Press release sobre agentes de IA para tareas específicas en aplicaciones enterprise, la proyección del 40% para 2026, frente a menos del 5% en 2025.
- Context Engineering: From Prompts to Corporate Multi-Agent Architecture, arXiv 2026, la fuente del principio de control sobre contexto, intención y especificaciones.
- FeatureBench: Benchmarking Agentic Coding for Complex Feature Development, arXiv 2026, la asimetría de benchmark entre correcciones de bugs y trabajo real de features.
- Anthropic, Effective context engineering for AI agents, guía de práctica sobre tratar el contexto como un artefacto de ingeniería.
- Model Context Protocol, el protocolo abierto detrás de la capa cold de retrieval y del soporte de MCP en 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 CAPA 01, CLOUD E INFRAESTRUCTURA

La fundación de cloud e infraestructura: dónde ejecutan los agentes, y a qué costo.

Antes de que un agente pueda saber algo u optimizar algo, tiene que ejecutarse en algún lugar. La Capa 01 es el sustrato: compute, red, identidad y entornos, dimensionados y gobernados para cargas agénticas, alimentando todas las capas superiores.

La brecha de deployment: por qué existe la capa de fundación

El State of AI in Business 2025 del MIT encontró que el 95% de los pilotos de GenAI enterprise no entregan valor medible, mientras el 78% de las organizaciones reporta usar IA en al menos una función. El embudo detrás de esa brecha es predecible: en un presupuesto enterprise típico de US$ 50 millones para IA, 42 POCs financiados se convierten en once pilotos internos, tres sistemas en producción y uno que genera ingresos o reducción de costos medible, el 2,4% de la inversión. Las organizaciones con un enfoque estructurado en las cuatro capas del stack alcanzaron una tasa de deployment del 62%, un orden de magnitud por encima de esa norma. La diferencia no es el modelo ni las herramientas. Es disciplina de capa, y la disciplina empieza por abajo.

La cadencia de deployment cuenta la misma historia desde las operaciones: cerca del 47% de las organizaciones despliega modelos de IA solo ocasionalmente y solo el 7% alcanza el deployment diario, la cadencia que la mejora continua exige. La brecha no es técnica. Es infraestructural: la mayoría de los equipos tiene compute, pero carece de la orquestación, la observabilidad, las fronteras de seguridad y la automatización que convierten un artefacto de modelo en un sistema de producción. La Capa 01 responde una pregunta: dónde ejecutan los agentes, y a qué costo. Para un equipo GitHub Copilot la pregunta es concreta. El agent mode en VS Code parece serverless, pero el Coding Agent ejecuta en entornos en la nube, los servidores MCP

corren como servicios reales, y todo lo que Copilot construye termina aterrizando en este sustrato.

El sustrato: Kubernetes, bloques de Azure y extensiones para agentes

Kubernetes es el runtime de facto para la IA en producción: dos de cada tres organizaciones que ejecutan IA a escala lo usan como plataforma primaria de orquestación. En Azure eso normalmente significa AKS para el runtime de los agentes, Microsoft Entra ID con managed identities para la identidad de workload, redes virtuales con private endpoints para que los agentes alcancen datos y herramientas por rutas privadas, y entornos separados de desarrollo, staging y producción para que un experimento nunca toque datos de producción. La consolidación se acelera: en marzo de 2026 la CNCF casi duplicó el número de plataformas Kubernetes AI certificadas, más del 70% de aumento en las certificaciones. Eso refleja dos hechos simultáneos: todos convergen en la misma plataforma, y el Kubernetes estándar necesita extensiones deliberadas para ejecutar cargas agénticas correctamente.

Las extensiones existen porque los agentes rompen las premisas de la infraestructura de ML tradicional. Un job de entrenamiento corre hasta terminar y un servicio de inferencia responde solicitudes stateless; un agente corre por duraciones desconocidas, ajusta recursos sobre la marcha, llama a servicios externos con latencia variable y ejecuta código que no escribió. El CNCF Kubernetes AI Conformance Program v1.35 nombra tres capacidades que hacen esto seguro: in-place pod resizing, para que los recursos se ajusten sin reinicio cuando un agente pasa de la planificación al uso intensivo de herramientas; workload-aware scheduling, que rompe las esperas circulares que atrapan a los schedulers FIFO cuando los agentes se coordinan; y sandboxing, aislamiento duro a nivel de SO o VM que resiste incluso ante código de agente hostil. Los Cloud Native Agentic Standards de marzo de 2026 agregan la tríada operacional: least privilege, observabilidad MELT con métricas, eventos, logs y traces, y tolerancia a fallos con retries y circuit breakers.

El pod de cuatro contenedores

A nivel de pod, una carga agéntica de producción ejecuta cuatro contenedores, no uno: el agent runtime con el bucle del modelo y la lógica de decisión; un sidecar MCP que termina conexiones MCP, aplica RBAC y rate limits de tokens y emite la pista de auditoría; un sidecar de contexto que sirve la capa de hot memory, el resolvedor de skills y el CONSTITUTION.md; y un sidecar de telemetría con el colector OpenTelemetry, decision spans y métricas MELT. Esa topología es la primera decisión de platform engineering que determina si el agente escalará.

Anatomía de costos, identidad y la frontera de protocolo

La anatomía de costos de un agente en producción sorprende a la mayoría de los equipos. En las líneas de base medidas en producción, el 42% es compute, la parte que todos miden. El 58% restante es invisible hasta la factura: 28% tokens de inferencia, 15% memoria y vector database, 10% red y proxies MCP y 5% observabilidad, casi siempre infrafinanciada. Un equipo que solo rastrea compute tiene una estimación de TCO desviada por un factor de dos. GitHub Copilot hace explícito el vector de tokens: desde el 1 de junio de 2026, el uso de modelos premium se factura en GitHub AI Credits, con 1 crédito = US$ 0,01, y cada sesión de agent mode o tarea del Coding Agent en el picker, sea con Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro o GPT-5.x, es una línea medida. Trate esos créditos como gasto de infraestructura, no como una curiosidad por asiento.

La identidad y la mediación completan la capa. Cada agente lleva un SPIFFE SVID distinto, una identidad criptográfica de workload, y cada permiso de recurso es una política en OPA: un agente resumidor obtiene acceso de lectura y una cuota diaria de 50K tokens; un agente de incidentes SRE obtiene acceso break-glass con approval gate en escrituras. Mezclar esos permisos en un solo agente, el antipatrón one-big-agent, significa que el primer incidente lo compromete todo. El acceso en sí fluye por el Model Context Protocol: los agentes nunca tocan bases de datos ni APIs directamente, hablan MCP con resource servers que aplican política, registran el acceso y limitan la tasa en la capa de protocolo. MCP gobierna más de 10.000 servidores activos, con 97 millones de descargas mensuales de SDK a inicios de 2026, bajo la Agentic AI Foundation en la Linux Foundation, con más de 170

organizaciones miembro en marzo de 2026, y su alcance es deliberadamente estrecho: la observabilidad pertenece a OpenTelemetry, la identidad de workload a SPIFFE/SPIRE, la coordinación entre agentes a A2A v1.0 y el control declarativo a GitOps. Los servidores MCP que usted configura hoy para el agent mode de Copilot hablan el mismo protocolo que hablarán sus agentes de producción.

> DEFINICIÓN
>
> El 42% del costo de un agente en producción es compute. El otro 58%, tokens de inferencia, memoria, red y observabilidad, permanece invisible hasta que llega la factura. Rastree los cinco vectores o su estimación de TCO estará desviada por un factor de dos.

De IaC a la operación autónoma: cómo la capa alimenta el stack

La infraestructura como código es lo mínimo; el patrón que hace confiable la IaC generada por LLM es el verificador en el loop. TerraFormer, de Georgia Tech con AWS, ejecuta terraform plan como feedback para el modelo y logra una mejora del 15,94% en el benchmark IaC-Eval, superando a modelos 50 veces más grandes. El mismo loop funciona en el agent mode de Copilot: codifique las convenciones de Terraform y de pipeline en .github/copilot-instructions.md, deje que el agente genere, valide con el plan, itere. La supply chain es el riesgo dominante en esta capa: de 295 GitHub Security Advisories que referencian componentes LLM entre enero de 2025 y enero de 2026, supply chain representa el 44%, excessive agency el 20% y prompt injection el 18%. Ningún escáner cubre todas las categorías, así que combine al menos dos, como zizmor más actionlint; con una mediana de 2,71 segundos por workflow, no hay excusa de rendimiento.

La trayectoria de la capa es la autonomía. Los adoptantes tempranos alcanzaron el 95% de aprovisionamiento automatizado en 2025, se vuelve estándar entre las empresas líderes en 2026, la gestión manual de infraestructura se convierte en un pasivo competitivo en 2027 y la operación totalmente autónoma es la línea de base enterprise en 2028. La arquitectura de referencia para ese estado final es la Cognitive Platform Engineering, cuatro planos sobre Kubernetes, Terraform y OPA,

data, intelligence, control y experience, cerrados por un loop Sense-Reason-Act: la plataforma percibe patrones de workload, razona sobre la configuración óptima y actúa antes de que alguien abra un ticket. Así es como la Capa 01 alimenta el stack: la ingeniería de plataforma hereda primitivas de identidad y cuota, la ingeniería de contexto hereda la mediación MCP y el sustrato de memoria, y la ingeniería de intención hereda la pista de auditoría que hace medible el drift. La infraestructura no es plomería, es política: cada decisión de scheduling, ruta de red y frontera de aislamiento aplica una elección sobre lo que está permitido, visible y seguro.

Referencias

Fuentes de la brecha de deployment, de los requisitos del sustrato y del stack de estándares.

- The GenAI Divide: State of AI in Business 2025, MIT NANDA, el dato del 95% de pilotos sin valor detrás de la brecha de deployment.
- CNCF Platforms Whitepaper, CNCF, la base de platform engineering sobre la que se apoyan las extensiones agénticas.
- Model Context Protocol, la frontera de protocolo entre los agentes y la infraestructura de la que dependen.
- Context Engineering: From Prompts to Corporate Multi-Agent Architecture, arXiv 2026, la arquitectura multiagente en capas que esta capa de fundación sostiene.
- Effective context engineering for AI agents, Anthropic, la capa superior a la que sirve este sustrato.

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 CAPA 02, INGENIERÍA DE PLATAFORMA

Ingeniería de plataforma: la capa de gobernanza de la entrega agéntica.

Los agentes no fallan en producción porque el modelo sea débil. Fallan porque nada gobierna lo que pueden tocar. Golden paths, guardrails, políticas y review gates son la capa que transforma el agent mode de GitHub Copilot de riesgo operacional en capacidad de entrega gobernada.

La ingeniería de plataforma se volvió la capa de gobernanza

El State of AI in Business 2025 del MIT constató que el 95% de los pilotos corporativos de GenAI no entrega valor medible. Eso es un problema de gobernanza, no de modelo, y la ingeniería de plataforma es donde la industria buscó la respuesta. La evaluación DORA de 2025 encontró que cerca del 90% de las empresas ya opera plataformas internas, y Gartner proyecta que el 80% de las grandes organizaciones de ingeniería de software tendrá equipos de plataforma dedicados hacia 2026. En el camino, el rol cambió: los equipos de plataforma dejaron de ser constructores de herramientas internas y se volvieron arquitectos de gobernanza.

El detonante es la llegada de los agentes como operadores. Cuando el agent mode de GitHub Copilot abre un pull request o genera Terraform, toma decisiones de infraestructura con autonomía y exige la gobernanza de un operador humano. Los resultados justifican la inversión: el equipo de plataforma de Spotify desplegó agentes que produjeron más de 1.500 pull requests fusionados, con ahorros de tiempo del 60% al 90% en migraciones, y Thomson Reuters entregó ganancias de productividad de 15x y un 70% de automatización en workflows críticos. El delta estructural es pequeño: un IDP agéntico tiene siete componentes, y solo dos difieren de un IDP estándar, una Agent API junto al portal humano y RBAC por agente junto a las políticas como código. Sin esos dos, la plataforma es buena para humanos e invisible para los agentes.

DEFINICIÓN

El principio operativo de este capítulo, capturado en KubeCon EU 2026: los agentes amplifican lo que es bueno en tu ecosistema y amplifican lo que es malo. Golden paths limpios y guardrails fuertes se ejecutan de forma impecable a escala; convenciones inconsistentes y controles de costo débiles se replican en miles de deployments. La higiene de plataforma es una precondición para los agentes, no una optimización.

Los cuatro pilares: golden paths, guardrails, safety nets y review gates

El framework de la CNCF identifica cuatro pilares que los equipos de plataforma deben implementar para gobernar agentes autónomos: golden paths, guardrails, safety nets y workflows de revisión manual. En el ecosistema de GitHub Copilot, cada pilar se materializa en un mecanismo concreto: custom instructions, validación de políticas en los pull requests, respuesta automatizada y review gates protegidos.

Golden paths: de la intención a la infraestructura

Los golden paths definen el camino bendecido para las tareas comunes: desplegar un servicio, aprovisionar una base de datos, solicitar secrets. Los caminos escritos a mano no sobreviven a la velocidad de los agentes, así que las plataformas maduras los vuelven generativos: el equipo de plataforma revisa y aprueba la plantilla del camino una vez, y los agentes la reutilizan miles de veces. En términos de Copilot, el golden path vive donde el agente ya mira, custom instructions del repositorio que codifican los estándares del equipo y plantillas expuestas vía MCP, para que el agent mode consuma el blueprint aprobado en lugar de improvisar uno. El equipo de plataforma deja de escribir Terraform todo el día y pasa a escribir esquemas de validación.

Guardrails: del escaneo reactivo al enforcement proactivo

Los guardrails detienen los malos resultados antes de producción. El modelo histórico, escáneres marcando violaciones después del deploy, no funciona a la

escala de los agentes. Policy-as-code invierte la lógica: el equipo de plataforma describe requisitos de alto nivel, cumplimiento de PCI-DSS, ningún secret hardcodeado, costo por servicio por debajo de US$ 10.000 por mes, y esos requisitos se convierten en políticas deterministas y ejecutables que validan el código generado por agentes antes de la ejecución. Una sola definición de política cubre miles de deployments conducidos por agentes, y los revisores humanos se enfocan en definir políticas y manejar excepciones, no en validar caso por caso.

Safety nets: respuesta autónoma

Las safety nets capturan lo que se escapa de los guardrails. Cuando aparece un nuevo CVE, la plataforma genera guardrails de runtime y los despliega en minutos, en lugar de las semanas que solía tomar un parche coordinado en cientos de deployments. La respuesta autónoma a incidentes detecta anomalías, dispara troubleshooting conducido por agentes y resuelve incidentes sin intervención humana, mientras el SRE predictivo actúa antes de que el usuario sienta el impacto.

Review gates: fricción estratégica

No todo debe automatizarse. Los equipos de plataforma definen gates de revisión manual para decisiones de alto riesgo: cambios en sistemas de pago, migraciones de bases de datos, expansiones de cuota de infraestructura. Esos gates desaceleran a los agentes a propósito. El auditor agent ensambla logs, diagramas de arquitectura y trazas de auditoría en un paquete de compliance completo con un score de riesgo, para que los revisores humanos dediquen su tiempo a decidir, no a recolectar información.

Agentes como ciudadanos de primera clase de la plataforma

Una plataforma madura trata al agente como una persona de usuario con identidad propia: cada agente lleva un SPIFFE SVID distinto, una identidad criptográfica de workload, y cada permiso de recurso es una política en OPA. Un agente de resumen obtiene acceso de lectura a la base de producción y una cuota diaria de tokens; un agente de incidentes de SRE obtiene acceso break-glass de emergencia con un approval gate en operaciones de escritura. Mezclar esos permisos en un solo

agente, el anti-patrón one-big-agent, significa que el primer incidente compromete todo. La misma disciplina responde a la tensión del vibe coding: un desarrollador le pide al agent mode de Copilot desplegar un servicio, recibe código de infraestructura de apariencia plausible y lo envía sin validar. La plataforma debe asumir que todo artefacto generado por agentes conlleva riesgo: validación de políticas obligatoria antes del merge, análisis de impacto de costo en cada deployment y rollback automático cuando se violan los umbrales.

MCP es la interfaz que hace que esto escale. En lugar de construir APIs a medida, el equipo de plataforma expone capacidades como servidores MCP, catálogo de servicios, aprovisionamiento de infraestructura, gestión de secrets, workflows de deployment, y mantiene un registro descubrible de ellos. Cuando llega un agente nuevo, descubre las capacidades a través del registro y opera dentro de las restricciones que encuentra: el agente hace su propio onboarding. Desde el 1 de junio de 2026, el uso premium de Copilot se factura en GitHub AI Credits, un crédito equivale a US$ 0,01, consumidos en proporción a los tokens procesados, lo que vuelve los presupuestos por agente exigibles en dólares. La ejecución rutinaria de golden paths puede correr en un modelo más barato del picker, como Haiku 4.5, mientras Claude Fable 5 u Opus 4.8 quedan reservados para el trabajo complejo que justifica su costo.

El reality check del code review

La narrativa de la industria sobre la revisión de código autónoma no coincide con los datos. Un estudio de 19.450 pull requests constató que los agentes de code review operando solos alcanzan una tasa de merge del 45,20% contra el 68,37% de los revisores humanos, y el 60,2% de los PRs revisados solo por agentes muestra una calidad de señal entre 0 y 30%. Un análisis de 278.790 conversaciones de revisión en 300 proyectos explica por qué ambos son complementarios: los comentarios de IA son densos en defectos, 29,6 tokens por línea contra 4,1 de los humanos, los humanos agregan entendimiento y transferencia de conocimiento, y el código de IA exige 11,8% más rondas de revisión. El pipeline se sigue directo: Copilot Code Review hace la primera pasada, un humano hace la segunda, y ninguna revisión de agente es jamás el único gate.

El merge tampoco es la línea de llegada. Un análisis de 1.210 PRs de corrección de bugs generados por agentes y fusionados, usando SonarQube, constató que el éxito en el merge no refleja la calidad post-merge, con un agente promediando 8,3 issues de análisis estático por PR fusionado. Un estudio de 33.000 PRs de cinco agentes mostró que documentación y CI tienen las mayores tasas de merge, performance y corrección de bugs las menores, y que el modo de falla más frecuente es el abandono del revisor: PRs confusos, demasiado grandes o mal descritos simplemente quedan abandonados. La respuesta de la plataforma es específica: quality gate post-merge obligatorio, límites de tamaño de PR y plantillas de descripción para código de agentes, y métricas de calidad post-merge en lugar de la tasa de merge. Las organizaciones que aciertan en esta capa reportan mejoras de productividad de 2 a 5x en los equipos de ingeniería.

> DEFINICIÓN
>
> Antes de escalar agentes, seis elementos forman la plataforma mínima viable: un catálogo de servicios completo, golden paths diseñados para workloads de IA, un registro de servidores MCP, RBAC por agente, observabilidad de las trayectorias de los agentes que capture intención y razonamiento, y gobernanza de costos por agente en AI Credits. La falta de cualquiera de los seis es un impedimento bloqueante para el deployment de agentes.

Referencias

Fuentes para la capa de ingeniería de plataforma, los cuatro pilares y la investigación de code review de este capítulo.

- CNCF Platforms Whitepaper, CNCF TAG App Delivery, la definición de referencia de plataformas internas, golden paths y capacidades de plataforma.
- Model Context Protocol, el protocolo abierto por el cual los agentes descubren las capacidades que un equipo de plataforma expone, incluido el agent mode de Copilot.
- The GenAI Divide: State of AI in Business 2025, MIT NANDA, el informe detrás de la cifra del 95 por ciento de pilotos sin valor medible.

- From Industry Claims to Empirical Reality: Code Review Agents, Kennesaw State y Quanta, 19.450 pull requests, la brecha de tasa de merge y calidad de señal entre agentes y humanos.
- Human-AI Synergy in Agentic Code Review, Queen's University, 278.790 conversaciones de revisión en 300 proyectos, por qué la revisión de IA y la humana son complementarias.
- Beyond Bug Fixes: Code Quality After Agentic Merges, University of Saskatchewan, 1.210 PRs fusionados analizados con SonarQube, por qué la tasa de merge no es un proxy de calidad.
- Where Do AI Coding Agents Fail?, Drexel University, 33.000 PRs de cinco agentes, el abandono del revisor como modo de falla 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 CAPA 03

Ingeniería de contexto: la curación de lo que el agente ve.

Archivos de instrucciones, contexto del repositorio, retrieval vía MCP y presupuestos de tokens: la disciplina de dar al agent mode de GitHub Copilot el menor conjunto de tokens de alta señal que maximiza el resultado.

Lo que el agente ve es una decisión de ingeniería

La ingeniería de contexto es la disciplina de estructurar el payload completo de información que un agente recibe en el momento de la inferencia. Donde el prompt engineering optimizaba una sola interacción, la ingeniería de contexto pregunta qué necesita saber realmente el agente, en qué orden, a qué costo, y cómo se gestiona a lo largo del tiempo. Cada solicitud que el agent mode de GitHub Copilot envía lleva archivos de instrucciones, fragmentos del repositorio, definiciones de tools, salidas de MCP e historial de la conversación, y desde el 1 de junio de 2026 todo eso se mide en GitHub AI Credits, con 1 crédito equivalente a US$ 0,01. Anthropic formula el objetivo con precisión: encontrar el menor conjunto posible de tokens de alta señal que maximice la probabilidad del resultado deseado.

Más contexto no es mejor contexto. Investigadores analizaron 18 LLMs líderes y documentaron una degradación severa e impredecible a medida que el contexto crece, el fenómeno conocido como context rot. En pruebas controladas, los mismos 4K tokens de material de alta relevancia superaron con frecuencia a 100K tokens de material de relevancia mixta. La implicación es directa: curar lo que el agente ve vale más que expandir lo que podría ver, y el punto óptimo de compresión debe medirse por modelo y por workload, no asumirse.

Cuatro componentes que deben mantener coherencia

La disciplina se apoya en cuatro piezas fuertemente acopladas. La composición selecciona y formatea lo que entra: hechos, instrucciones, definiciones de tools,

ejemplos. El retrieval suministra contexto en runtime vía búsqueda semántica y salidas dinámicas de tools. La memoria gestiona el estado a lo largo del tiempo. Los protocolos estandarizan cómo los agentes reciben y publican contexto, principalmente MCP. En términos de Copilot, la composición son sus archivos de instrucciones, el retrieval es la búsqueda en el workspace más los servidores MCP, la memoria es lo que persiste entre sesiones del agente, y los protocolos son la propia configuración de MCP.

Los archivos de instrucciones son infraestructura, no prompts

El archivo .github/copilot-instructions.md se carga en cada solicitud, lo que convierte su tamaño en una factura recurrente. Haga las cuentas: un archivo de instrucciones de 2.000 tokens, 1.000 desarrolladores, 10 solicitudes por día cada uno. Son 20 millones de tokens por día antes de una sola línea de trabajo real, cerca de US$ 60 por día a US$ 0,003 por 1.000 tokens de input, o 6.000 AI Credits. Los archivos .instructions.md con alcance por ruta vía applyTo recortan ese costo en aproximadamente un 68%: una instrucción sobre manejar fallos de pago se carga solo para trabajo en el subsistema de pagos. Combinado con skills de carga diferida, el mismo día baja a 6,4 millones de tokens, cerca de US$ 19,20, y la diferencia acumula unos US$ 15 mil por año en esta escala modesta.

La evidencia más fuerte es sobre quién escribe estos archivos. Un experimento controlado en 47 repositorios internos mostró que los archivos AGENTS.md curados por humanos redujeron el runtime en 28,64% y el consumo de tokens en 16,58%. Los equivalentes generados por LLM añadieron 3,02% de overhead, peor que no tener archivo alguno: las especificaciones generadas automáticamente documentan lo obvio y omiten lo crítico, el conocimiento tácito del equipo que solo los humanos identifican. La conclusión: los archivos de instrucciones son documentación de ingeniería, con peer review, versionado, tratamiento de código de producción, responsable nombrado y cadencia de curación.

DEFINICIÓN

Un archivo de contexto tiene exactamente dos estados: señal curada o ruido en acumulación. La prueba es la propiedad. Si nadie revisa los cambios al copilot-instructions.md en pull requests, el archivo está derivando hacia el ruido, y cada desarrollador paga esa deriva en AI Credits en cada solicitud.

Capas de memoria y disciplina de retrieval vía MCP

Un agente bien diseñado carga por capa, no por defecto. La memoria hot, cerca de 660 tokens, está siempre en contexto: identidad del agente, convenciones inviolables, hooks de seguridad, actualizada solo en deploy. La memoria warm, cerca de 2.000 tokens por skill activada, se carga bajo demanda: instrucciones especialistas, AGENTS.md por repositorio, esquemas de dominio, con curación semanal de los tech leads. La memoria cold, cerca de 1.500 tokens por query, nunca se precarga: la base de conocimiento corporativa, datos de producción y código, recuperados vía servidores MCP con control de acceso por agente. El resultado se mantiene en torno a 4.160 tokens por invocación, mientras un agente mal diseñado vuelca todo en la memoria hot y paga 30.000 tokens por una tarea trivial.

El tiering no es teórico. Un deployment de producción aplicó esta arquitectura a un backend C# de 108.247 líneas durante 12 semanas, cubriendo 283 sesiones de agente y 2.801 prompts de usuario. Los tokens por tarea cayeron 32%, de 18 mil a 12,2 mil por invocación, la latencia p95 cayó 47%, de 9,2 a 4,9 segundos, y la completación de tareas subió 18%. Las skills aplican el mismo principio con lazy loading: las descripciones de 50 a 100 tokens quedan visibles, el cuerpo completo solo se carga en la invocación; con cerca del 70% de las skills instaladas sin uso en una sesión, solo eso recorta el costo efectivo de contexto en 70%.

El contexto de MCP tiene smells; audítelo

Cada servidor MCP habilitado en Copilot aporta descripciones de tools al contexto, y esas descripciones sufren lo que los investigadores llaman tool description smells. Una tool descrita como 'procesa transacciones' invita al modelo a alucinar los

detalles ausentes; una spec sin documentación de errores falla en silencio cuando la tool real lanza una excepción. Auditar descripciones en busca de vaguedad, restricciones de parámetros ausentes y manejo de errores faltante es ingeniería de contexto poco glamorosa y esencial. El ecosistema de skills muestra la negligencia a escala: de 55.315 skills públicas encuestadas, el 26,4% carece de descripción de enrutamiento, lo que las vuelve indescubribles, y más del 60% del contenido de los cuerpos es relleno no accionable. Quitar ese ruido comprimió las descripciones en 48% y los cuerpos en 39%, mejorando la calidad funcional en 2,8%.

Presupuestos de contexto e higiene, medidos en AI Credits

Trate el presupuesto de contexto como un presupuesto de latencia: un número explícito por invocación, seguido en AI Credits. Presupuestar, sin embargo, no es encoger a ciegas. La paradoja de la compresión está documentada: comprimir el contexto agresivamente redujo los tokens de input en 17% y aumentó el costo total de la sesión en 67%, porque el modelo quemó muchos más tokens de output reconstruyendo significado. La idea rectora es la densidad semántica. Los tokens que llevan información real, nombres descriptivos, anotaciones de tipo, docstrings bien escritas, son inversiones; el boilerplate y la ceremonia son puro desperdicio. Los archivos de instrucciones deben usar lenguaje natural rico y claro en lugar de abreviaturas, y los recortes deben eliminar ruido, nunca señal.

La higiene significa resistir dos modos de fallo a lo largo del tiempo. El brevity bias descarta insight específico de dominio en nombre de la compresión. El context collapse es peor: en una ejecución documentada, un contexto de agente con 18.282 tokens lograba 66,7% de exactitud; reescrito por brevedad, cayó a 122 tokens y 57,1%, por debajo de la línea de base sin contexto de 63,7%. La alternativa es la curación deliberada: el enfoque ACE, generación, reflexión y curación sobre un contexto que evoluciona, entrega +10,6% de exactitud sobre líneas de base de contexto estático en benchmarks de agente y +8,6% en tareas de análisis financiero.

Ajuste la profundidad de contexto al modelo en el picker

Emparejar workload con modelo es ingeniería de contexto, y el model picker de Copilot es donde ocurre. El trabajo rápido de acción, ediciones y ejecuciones de

tools, corre ligero en Sonnet 5 o Haiku 4.5: optimice latencia, no profundidad de contexto. El razonamiento extendido sobre estado complejo, revisiones de compliance, decisiones de arquitectura, pertenece a Claude Fable 5 u Opus 4.8, donde 50K a 100K tokens por invocación pueden ser un gasto justificado. La compactación y los handoffs a subagentes son territorio de Haiku 4.5: contexto mínimo, especificidad máxima. El parsing multimodal de PDFs, diagramas y tablas pide un modelo con visión, como Gemini 3.1 Pro o un GPT-5.x. Un workload de razonamiento en un modelo rápido entrega de menos; un workload de acción en Opus 4.8 sobreaprovisiona AI Credits.

Referencias

Fuentes para la disciplina de curación, las capas de memoria y la economía de tokens.

- Effective Context Engineering for AI Agents, Anthropic, el encuadre del menor conjunto de tokens de alta señal.
- Agentic Context Engineering, arXiv, contextos que evolucionan, brevity bias y context collapse.
- Codified Context: Infrastructure for AI Agents in a Complex Codebase, arXiv, los números de producción del three-tier en un backend C# de 108 mil líneas.
- A Survey of Context Engineering for Large Language Models, arXiv, el mapa de investigación de composición, retrieval, memoria y protocolos.
- Model Context Protocol, MCP, el protocolo abierto detrás del retrieval de tools y datos en el agent mode de 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 CAPA 04, INGENIERÍA DE INTENCIÓN

Ingeniería de intención, expresar qué construir.

El contexto le dice al agente qué es verdad. La intención le dice qué debe querer. Este capítulo convierte el propósito organizacional en artefactos que los agentes de GitHub Copilot pueden leer, verificar y respetar.

Contexto sin intención optimiza lo incorrecto

El caso Klarna, del tercer trimestre de 2025, muestra qué pasa cuando se salta esta capa. El agente tenía todo lo que la ingeniería de contexto puede ofrecer, de datos detallados de clientes a reglas de precios e historial de ejecución, y aun así en semanas empezó a optimizar métricas que la organización nunca quiso priorizar; cuando alguien lo notó, el desalineamiento ya había producido costos operativos y reputacionales sustanciales. La ingeniería de contexto hace más precisas las decisiones del agente para un objetivo dado, pero no dice cuál debe ser el objetivo. Un agente bien contextualizado sin intención es un piloto con datos meteorológicos y especificaciones de combustible, pero sin plan de vuelo.

La pirámide de cuatro niveles

La jerarquía de la fuente tiene cuatro niveles. Nivel 1, prompt engineering: una instrucción para un modelo en una interacción; no escala entre sesiones ni equipos. Nivel 2, ingeniería de contexto: cargas de información, capas de memoria y retrieval vía MCP, tema del capítulo anterior. Nivel 3, ingeniería de intención: objetivos, valores y jerarquías de trade-off, respondiendo qué debe lograr el sistema y qué restricciones son innegociables. Nivel 4, ingeniería de especificación: un corpus legible por máquina de políticas, estándares de calidad y restricciones de implementación para una operación multiagente coherente a escala. La mayoría de las organizaciones opera en los niveles 1 y 2; la distancia hasta el 3 y el 4 es donde se acumula el intent debt.

Intent debt, una deuda de claridad

El intent debt no es deuda de esfuerzo, es deuda de claridad: se acumula cuando los objetivos, las restricciones y el razonamiento detrás de ellos viven en un hilo de Slack de hace ocho meses o en la cabeza de un solo ingeniero, y no en artefactos duraderos. Tres mecanismos lo convierten en riesgo: concentración de conocimiento, cuando la salida de personas deja restricciones sin documentar; drift de optimización, cuando objetivos no declarados empujan a humanos y agentes hacia proxies medibles; y ausencia compuesta, cuando cada nuevo agente agrega divergencia sin fuerza correctiva. Cuando los agentes toman decisiones de consecuencia sin supervisión, esa deuda es riesgo operacional.

Tres artefactos, y prompts como contratos

La ingeniería de intención se materializa en tres artefactos de repositorio. CONSTITUTION.md es un documento de principios y valores innegociables, escrito para humanos y estructurado para que las máquinas lo apliquen. Cinco secciones obligatorias: Identity, con alcance y límites; Values and trade-offs, jerarquía ordenada del tipo prefiere X a Y, siempre; Hard constraints, lo que el agente nunca hace; Protocols para situaciones recurrentes; y Escalation hooks, cuándo el agente se detiene y llama a un humano. Unas 200 líneas, con commit en Git.

SPECIFICATION.md en notación EARS

La especificación traduce los principios constitucionales en criterios de aceptación verificables por máquina usando EARS, el Easy Approach to Requirements Syntax de ISO/IEC/IEEE 29148, creado para Rolls-Royce en 2009. El requisito vago de resiliencia ante fallos de red se convierte en: cuando una llamada de red falle 3 veces en 60 segundos, el sistema deberá abrir el circuit breaker; mientras el circuito esté abierto, deberá servir respuestas en caché durante hasta 30 segundos. Los requisitos dejan de ser prosa y se vuelven condiciones que el agente verifica antes de ejecutar.

IMPLEMENTATION_PLAN.md y el repositorio Copilot

El plan divide el trabajo en tareas atómicas con marcadores [P] para lo que puede correr en paralelo y sirve de frontera de control: el agente sabe exactamente qué

puede modificar. En GitHub Copilot, el trío vive en el repositorio como contratos versionados: las custom instructions en .github/copilot-instructions.md llevan los principios constitucionales a cada solicitud, los prompt files reutilizables en .github/prompts/ codifican los rituales de especificación, y un agente personalizado lee CONSTITUTION, SPECIFICATION e IMPLEMENTATION_PLAN antes de escribir código. Un prompt que importa no es texto desechable de chat; es un archivo revisado, con dueño e historial de diffs.

> DEFINICIÓN
>
> Un agente bien contextualizado sin intención escrita es una flecha con mira excelente y ningún blanco. La ingeniería de intención aporta el mapa, el destino y los trade-offs que el agente está autorizado a hacer.

Spec-driven development, el flujo ejecutable

El Spec-Driven Development organiza el trabajo asistido por agentes en fases con gates de aprobación: humano y planner redactan CONSTITUTION.md y SPECIFICATION.md en modo Ask, sin escritura de archivos; la especificación se refina en un IMPLEMENTATION_PLAN.md; sigue el gate de aprobación humana; y solo tras el visto bueno explícito el agente entra en modo Agent, con escrituras limitadas al plan y code review controlando el merge. El gate es una frontera técnica aplicada por la capa de orquestación, no una convención blanda, y la investigación sobre IA constitucional reporta que el SDD reduce los defectos de seguridad en 73% frente al desarrollo asistido por IA sin restricciones.

El gap planner-coder

Un estudio de fallos en sistemas multiagente encontró que el 75,3% se origina en el gap entre planner y coder: especificaciones ambiguas o incompletas se convierten en código inseguro o incorrecto. El gap es además un vector de seguridad: un atacante que influye en el planner vía prompt injection puede hacer que el coder genere código vulnerable sin tocarlo nunca. El SPECIFICATION.md es la interfaz entre planificación y codificación; hacerlo explícito, interpretable por máquina y revisado por humanos cierra la superficie donde nacen tres cuartos de los fallos.

Modos, modelos y AI Credits

El flujo mapea al selector de modelos de Copilot. Planifique en modo Ask con un modelo premium como Claude Opus 4.8, o Claude Fable 5 para los trade-offs de arquitectura más difíciles; implemente en modo Agent con Claude Sonnet 5 como modelo de trabajo. Planificar hace pocas llamadas y cuesta pocos GitHub AI Credits, la unidad de facturación desde el 1 de junio de 2026, donde 1 crédito equivale a $0,01; ejecutar hace muchas. Dos disciplinas protegen intención y presupuesto: los hooks que bloquean escrituras fuera del plan corren en la capa de orquestación y consumen cero créditos, y el extended thinking se queda en la planificación, porque en implementación iterativa agrega 43% al costo de tokens y degrada la calidad en 30% en 4.018 trayectorias de SWE-Bench.

Verifique intención, no sintaxis

Los tests generados por agentes no verifican intención. En un estudio con seis LLMs en SWE-bench Verified, el 74,4% de los traces del agente más fuerte incluía tests escritos por el propio agente, pero funcionan como feedback observacional, más cercanos a print statements que a assertions; GPT-5.2 alcanzó 71,8% de éxito escribiendo casi ningún test. Las preocupaciones deben mantenerse separadas: el SPECIFICATION.md define qué debe testearse, y un humano revisa si los tests generados realmente lo testean. Bajo cambios semánticos, la tasa de aprobación de tests generados por LLM cae de 79% a 66% y la cobertura de branches de 76% a 60%; después de una refactorización significativa, regenere y revise en lugar de confiar en ellos.

Las métricas del drift

Cuatro señales revelan el drift temprano. La tasa de scope creep por sesión mide con qué frecuencia el agente propone trabajo fuera del plan. El porcentaje de ediciones fuera del IMPLEMENTATION_PLAN.md rastrea lo que revela el enforcement de los hooks y debe tender a cero. El tiempo de revisión humana por pull request generado por agente es un indicador temprano de planes subespecificados. Y los indicadores de deuda cognitiva, como el tiempo de onboarding, cuántas personas pueden explicar por qué existe una restricción y la reticencia a modificar ciertos archivos, exponen intención que nunca se escribió.

Dos palancas que los artefactos no reemplazan

Un estudio de 9.374 trayectorias en 19 agentes, 8 frameworks y 14 LLMs encontró que el modelo determina el resultado más que el framework, y que las trayectorias exitosas siguen la secuencia entender, reproducir, corregir, verificar. La elección en el selector de Copilot es una decisión de gobernanza de primer orden, y un hook que exige entendimiento declarado antes de editar elimina una clase entera de fallos. La segunda palanca es humana: la IA durante el aprendizaje reduce en 17% la adquisición de habilidades específicas de biblioteca, y los patrones que la preservan, indagación conceptual, código con explicación y generación seguida de comprensión, protegen la habilidad de la que depende la ingeniería de intención: escribir y evaluar especificaciones.

Referencias

Fuentes de la pirámide de cuatro niveles, los artefactos de intención, el flujo SDD y los datos de verificación.

- Context Engineering: From Prompts to Corporate Multi-Agent Architecture, arXiv 2026, la jerarquía de cuatro niveles de prompts a ingeniería de especificación.
- Storey (2026), arXiv:2603.22106, intent debt, sus tres mecanismos de riesgo y las métricas de drift.
- Specification-Driven Development for AI-Assisted Software Engineering, SANER 2026, la estructura de artefactos CONSTITUTION, SPECIFICATION y EARS.
- Constitutional AI reduces security defects in specification-driven development, la reducción de 73% frente al desarrollo asistido por IA sin restricciones.
- Planner-Coder Gap in Multi-Agent Systems, el 75,3% de los fallos multiagente nacidos entre planner y coder.
- Rethinking the Value of Agent-Generated Tests, seis LLMs en SWE-bench Verified, tests 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 INTEGRACIÓN Y HARNESS

Integración y harness engineering: opera el stack como un único sistema.

Las cuatro capas del Context Platform Stack solo generan retorno cuando corren como un único loop de entrega, y ese loop solo permanece confiable cuando un harness de guías, sensores y evals envuelve a cada agente. Este capítulo arma el loop, nombra el harness y cierra con una secuencia de adopción de 90 días.

Un loop de entrega, cuatro capas

El Context Platform Stack no es un proceso secuencial, es un grafo de dependencias. Cloud e infraestructura responden dónde ejecutan los agentes y a qué costo. La ingeniería de plataforma agrega la gobernanza, el internal developer platform, golden paths, RBAC y controles de costo que responden qué pueden hacer los agentes y quién puede desplegarlos. La ingeniería de contexto agrega la cognición, skills, capas de memoria y servidores MCP que responden qué sabe el agente. La ingeniería de intención agrega la estrategia, CONSTITUTION.md y especificaciones que responden qué debe optimizar el agente. El flujo es bidireccional: la intención restringe el contexto, el contexto determina qué capacidades de plataforma importan, y las decisiones de plataforma restringen la infraestructura. La mayoría de los fallos ocurre en esos loops de feedback: infraestructura construida sin intención, contexto crecido sin gobernanza, agentes lanzados sin especificación.

Saltarse el loop lleva al cementerio de agentes, el repositorio de prototipos abandonados que casi toda gran empresa acumula. El State of AI in Business 2025 del MIT encontró que el 95% de los pilotos de GenAI enterprise no entregan valor medible, mientras que las organizaciones que trabajan las cuatro capas como un sistema alcanzaron una tasa de deployment del 62% en los engagements detrás de este playbook. Los benchmarks explican parte de la brecha: Claude Opus 4.5

alcanza el 74,4% en SWE-bench, correcciones de bugs, pero solo el 11,0% en FeatureBench, el benchmark de la ICLR 2026 con 200 tareas a nivel de feature, cerca de 7 veces más difíciles que las correcciones de bugs. Un POC que solo demuestra correcciones de bugs opera en la zona fácil y sobreestima la capacidad en producción.

> DEFINICIÓN
>
> Cinco alertas tempranas de que vas camino al cementerio: agentes sin SLOs ni seguimiento diario de KPIs, ningún responsable nombrado por la evolución del agente, ningún golden path para el próximo agente, contexto mantenido por una sola persona e intent drift que pasa desapercibido por más de dos semanas. Arregla la disciplina antes de escalar la flota.

Enruta modelos por fase del SDLC y paga en AI Credits

La elección de modelo es específica por tarea y por fase, no global, y el model picker de GitHub Copilot es donde aterriza la disciplina. Especificación y arquitectura exigen descomposición bajo incertidumbre, así que se enrutan a un modelo premium de razonamiento, Claude Opus 4.8 o Claude Fable 5 con extended thinking, en modo ask. La implementación de features de 3 a 10 archivos se enruta a Claude Sonnet 5 en agent mode con hooks habilitados. Docstrings, mensajes de commit y resúmenes de PR se enrutan a Claude Haiku 4.5. Gemini 3.1 Pro y la familia GPT-5.x cubren los tiers equivalentes para equipos estandarizados en otros proveedores. La investigación detrás de la división es específica: extended thinking cuesta 43% más y degrada la calidad en 30% en tareas iterativas de implementación, así que el razonamiento premium dentro de loops apretados de código es a la vez desperdicio y peor.

Desde el 1 de junio de 2026 esta tabla de ruteo también es un instrumento de presupuesto, porque cada llamada premium en Copilot se cobra en GitHub AI Credits y un crédito corresponde a un centavo de dólar de uso de modelo a precio de lista. Un equipo que reserva Opus 4.8 para especificación y revisión, corre la implementación en Sonnet 5 y empuja los resúmenes a Haiku 4.5 paga por

profundidad cognitiva exactamente donde compra calidad. El mapa de madurez dice por dónde empezar: la etapa 0 es el caos y la etapa 4 es intent driven, con todas las capas activas. Tu etapa es el mínimo de las cuatro capas, no la mejor de ellas: métricas DORA sin curación de AGENTS.md es etapa 1, y cincuenta agentes sin métricas de intención sigue siendo etapa 2. Cada transición toma uno o dos trimestres.

El harness: guías, sensores y evals

La fórmula que cristalizó a inicios de 2026 es compacta: un agente es un modelo más un harness. El modelo aporta el razonamiento; el harness es todo lo que lo rodea, herramientas, memoria, estado, gates y validación. Mitchell Hashimoto acuñó engineer the harness el 5 de febrero. El 11 de febrero, Ryan Lopopolo, de OpenAI, reportó tres ingenieros entregando cerca de un millón de líneas de código de producción con cero código escrito por humanos y nombró la disciplina harness engineering. El 17 de febrero, LangChain saltó del puesto 30 al puesto 5 en Terminal Bench 2.0 cambiando solo el harness, con el modelo constante. El 3 de abril, Microsoft Agent Framework 1.0 llegó a GA con una capa literalmente llamada Agent Harness. El modelo es el componente intercambiable; el harness es el diferencial.

En marzo de 2026, Martin Fowler y Birgitta Böckeler organizaron el harness en guías, que dirigen al agente, y sensores, que lo verifican. El patrón de tres agentes de Anthropic muestra por qué importa la separación: un Planner expande el prompt en una especificación, un Generator implementa en sprints y un Evaluator navega por la aplicación como un usuario real antes de cerrar cualquier sprint, con una definición de terminado compartida y acordada de antemano. La investigación atribuye el 75,3% de los fallos de agentes a la interfaz planner coder, y este patrón hace esa interfaz explícita y verificada.

Las guías dirigen al agente

En GitHub Copilot las guías son archivos concretos: .github/copilot-instructions.md como el mapa del repositorio, archivos .instructions.md con alcance por ruta, archivos .prompt.md reutilizables para tareas recurrentes y configuraciones de servidores MCP que definen qué herramientas existen. Mantén el mapa en torno a 100 líneas apuntando a docs más profundas, un índice, no una enciclopedia. Y

cúralo a mano: en 47 repositorios internos, archivos AGENTS.md curados por humanos recortaron el runtime en 28,64% y el consumo de tokens en 16,58%, mientras que las versiones generadas por LLM agregaron 3,02% de overhead.

Los sensores verifican la salida

Los sensores determinísticos son baratos y atrapan la mayoría de los fallos: linters personalizados que imponen la dirección de dependencias, type checkers, tests estructurales, pre commit hooks, gates de CI que bloquean merges y detectores de drift de schema. Los sensores inferenciales, revisión LLM as judge y la automatización de code review de Copilot en pull requests, atrapan el juicio semántico que los linters no alcanzan. La lección de LangChain: muchas veces el problema no es el modelo, es el sensor que falta; la detección de loops y la autoverificación obligatoria impulsaron su salto en Terminal Bench. Los evals cierran el conjunto: un golden set de prompts representativos corre en CI en cada cambio del harness y bloquea regresiones antes del ship.

La secuencia de adopción de 90 días

La definición de Hashimoto es la disciplina operativa: cada vez que el agente comete un error, ingenia el entorno para que ese error se vuelva estructuralmente imposible de repetir. El loop tiene seis pasos: observa el fallo en una trayectoria capturada, clasifica la causa raíz, decide el tipo de corrección, un nuevo sensor, herramienta, actualización de instrucciones, gate o subagente, implementa la corrección, agrega un test de regresión y audita las trayectorias futuras para confirmar que la corrección se sostuvo. Las trayectorias capturadas desde el primer día se convierten en un activo que se compone: cada sesión produce datos de entrenamiento, datos de eval y documentación de edge cases.

La secuencia en sí cabe en un trimestre. Días 1 a 30: elige un caso de uso con criterio de éxito verificable, escribe el .github/copilot-instructions.md, instrumenta trayectorias, define una política mínima de aprobación y corre el primer experimento contra un baseline medido. Días 31 a 60: planta sensores determinísticos, codifica las tres a cinco reglas arquitectónicas más importantes como mecanismos y agrega autoverificación más revisión LLM as judge en pull requests. Días 61 a 90: configura identidad y auditoría para los agentes en uso, replica el patrón en un segundo

repositorio y mide contra el baseline. Mide también más allá de la velocidad: un estudio con 2.989 desarrolladores identificó seis dimensiones que las métricas al estilo DORA no capturan, autosuficiencia, carga cognitiva, finalización de tareas, calidad de la revisión por pares, desarrollo de expertise y ownership del código.

Referencias

Fuentes primarias para el loop de integración y la disciplina de harness engineering.

- My AI Adoption Journey, Mitchell Hashimoto, 5 de febrero de 2026, el post que acuñó engineer the harness.
- Harness engineering: leveraging Codex in an agent-first world, Ryan Lopopolo, OpenAI Engineering Blog, 11 de febrero de 2026.
- Improving Deep Agents with harness engineering, LangChain, 17 de febrero de 2026, el estudio de caso de Terminal Bench 2.0.
- Harness engineering for coding agent users, Birgitta Böckeler, martinfowler.com, marzo de 2026, la taxonomía de guías y sensores.
- Harness design for long-running application development, Anthropic Engineering, marzo de 2026, el patrón Planner, Generator, Evaluator.
- Microsoft Agent Framework Version 1.0, Microsoft, abril de 2026, el release GA con la capa Agent Harness.
- FeatureBench: Benchmarking Agentic Coding for Complex Feature Development, arXiv 2026, el benchmark detrás de la brecha entre feature work y corrección de bugs.

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