THE FULL AGENTIC ENGINEERING PLATFORM · PLAYBOOK

Seis disciplinas deciden si los agentes entregan o van al cementerio.

La mayoría de los pilotos agénticos muere en producción, no por falta de ambición o presupuesto, sino por falta de ingeniería. Este playbook presenta la pila completa en seis disciplinas, Cloud, Platform, Context, Intent, Harness y Token Economy, bajo la lente de GitHub Copilot. Desde el 1 de junio de 2026, cuando GitHub AI Credits hizo el costo por trabajo realizado, esa disciplina paga entre un 30 y un 70 por ciento menos en la factura.

AUTORA

Paula Silva

CARGO

AI-Native Software Engineer

CONTACTO

in/paulanunes

TIEMPO DE LECTURA

~ 45 minutos, de punta a punta

6

DISCIPLINAS DE INGENIERÍA

6

CAPÍTULOS

8

PATRONES DE OPTIMIZACIÓN

30-70%

REDUCCIÓN DE COSTO DE TOKENS

CAPÍTULOS

Chapter 00

El cementerio de agentes y las seis disciplinas

Por qué los pilotos agénticos mueren en producción, las tres deudas que se acumulan, la pila de seis disciplinas en secuencia y el cambio de GitHub AI Credits de junio de 2026 que hace que la disciplina se pague.

Fundaciones →

Chapter 01

Fundación cloud e ingeniería de plataforma

Kubernetes para agentes de larga vida, MCP como capa de protocolo, golden paths e identidad SPIFFE, y la arquitectura CPE de cuatro planos con seguridad de cadena de suministro.

Sustrato →

Chapter 02

Ingeniería de contexto

La curva de utilidad cóncava y el context rot, niveles de memoria con copilot-instructions.md y MCP, seis técnicas que se componen y el prompt caching.

Cognición →

Chapter 03

Intent engineering

La deuda de intención y el caso Klarna, los artefactos del Spec Kit, el Spec-Driven Development con compuerta humana y hooks de costo cero, y la brecha planner-coder.

Estrategia →

Chapter 04

Harness engineering

Agente es modelo más harness, la taxonomía de guías y sensores, cuatro patrones de producción, el playbook de siete pasos y dos caminos sobre una fundación Microsoft.

Modelo operativo →

Chapter 05

Economía de tokens, FinOps y adopción

Enrutamiento de modelos en el selector de Copilot facturado en AI Credits, FinOps para IA, ocho patrones de optimización, Azure AI Foundry y la hoja de ruta de 30, 60, 90 días.

Financiamiento →

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

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

00 EL PROBLEMA DEL CEMENTERIO

El cementerio de agentes y las seis disciplinas.

La mayoría de los pilotos agénticos nunca llega a producción. La brecha no es de ambición ni de presupuesto. Es de ingeniería. Un agente que funciona en el demo muere en producción porque el equipo se saltó capas: cloud, plataforma, contexto, intención, modelo operativo y economía de tokens. Este capítulo nombra por qué mueren los pilotos, presenta las seis disciplinas en secuencia y enuncia los dos hechos que hacen que la disciplina se pague sola.

Por qué los pilotos mueren en producción: tres deudas que se acumulan

Los pilotos agénticos no mueren porque el modelo sea débil. Mueren porque el equipo entregó sin una fundación de ingeniería coherente, y tres deudas se acumulan por debajo del demo. La primera es la deuda técnica, la conocida: parches, dependencias frágiles, tests quebradizos. Un estudio a gran escala de Liu y colegas analizó 304.362 commits generados por IA en 6.275 repositorios y encontró proporciones mayores de deuda de requisitos y de tests que en el código escrito por humanos. La asistencia de IA no elimina la deuda. Cambia la forma de la deuda.

La segunda deuda es la cognitiva, y es más peligrosa porque la revisión no la atrapa. Los desarrolladores aceptan código generado sin entender por qué funciona. Storey lo llama rendición cognitiva. La base de código se vuelve conocimiento huérfano: corre, pero nadie en el equipo puede modificarla con seguridad porque nadie entiende el razonamiento detrás. La ganancia de productividad del primer día se convierte en un pasivo de mantenimiento en el día noventa.

Deuda de intención: la falla que parece éxito

La tercera deuda es la de intención, la más nueva y la más insidiosa. El agente fue construido para la métrica X, bajo la restricción Y, con la premisa oculta Z. Cuando

las condiciones de producción cambian, el agente sigue optimizando para lo equivocado. No está roto. Simplemente nunca se especificó por completo. La deuda de intención es la razón por la que un agente puede pasar todos los tests y aun así hacer el trabajo equivocado, y por eso la especificación, no el código, es el artefacto que hay que revisar primero.

La pila de seis disciplinas, en secuencia

Una pila que sobrevive en producción cubre seis disciplinas, y el orden importa. Cloud va primero: Kubernetes y el Model Context Protocol dan a los agentes un sustrato que mantiene estado a lo largo de sesiones largas, en vez de tratar cada llamada como un request web sin estado. Platform va segundo: golden paths, una plataforma interna de desarrollo y templates de Backstage integran controles de seguridad y FinOps en cada agente por defecto, para que la gobernanza no sea un ejercicio de papel. Context va tercero: el conjunto más pequeño de tokens de alta señal que el agente necesita, ensamblado por memoria, MCP y recuperación, y precomputado en CI en vez de reconstruido en cada tarea.

Intent va cuarto: el Spec Kit hace legible el objetivo mediante una constitución, una especificación y un plan de implementación, para que el agente optimice para el resultado que de verdad quieres. El modelo operativo va quinto, la disciplina Harness: cadencia, propiedad, evidencia y revisión que convierten el framework de diapositivas en software entregado, por Three Horizons para equipos alineados con Microsoft o por Open Horizons para el stack CNCF. La economía de tokens va sexta: FinOps para agentes, con presupuestos por sesión, agente y repositorio, y Azure AI Foundry para el enrutamiento de modelos. Sáltate cualquier capa y el piloto entra al cementerio.

El hecho de junio de 2026: la facturación pasó de intención a trabajo realizado

El 1 de junio de 2026, GitHub Copilot dejó de cobrar por request. Los Premium Request Units se retiraron. GitHub AI Credits tomó su lugar, y la conversión es simple: 1 AI Credit equivale a US$0,01. Los tokens de input, output y cached se contabilizan ahora a la tarifa de API publicada de cada modelo en el Copilot picker,

desde Haiku 4.5 para trabajo trivial hasta Sonnet 5 para código y Opus 4.8 para planificación, junto a Claude Fable 5, Gemini 3.1 Pro y los modelos GPT-5.x. Code completions y Next Edit Suggestions siguen incluidos en el plan y no consumen créditos.

El cambio es estructural, no cosmético. Bajo los Premium Request Units, el costo era por intención: un request, un cargo, sin importar cuánto trabajo disparaba. Bajo GitHub AI Credits, el costo es por trabajo realizado. Una sesión en agent mode que hace fan-out por un repositorio cuesta lo que de verdad consume. Eso convierte la disciplina de tokens en un control financiero, no en un lujo. Cada técnica de este playbook, del contexto precomputado en CI, al enrutamiento de modelos en el Copilot picker, al prompt caching, aparece ahora directo en la factura.

Los números que justifican el esfuerzo

El caso de la plataforma se apoya en evidencia, no en un deck de proveedor. Deloitte encuestó a 3.235 respondientes en 24 países y encontró que el 75% de las empresas planea IA agéntica en dos años, pero solo el 34% reporta usar IA para transformar profundamente el negocio. La brecha entre planear y transformar es deuda de ingeniería, y es predecible. Un estudio causal de Carnegie Mellon con el dataset AIDev encontró que los agentes de codificación sin gobernanza añaden un 39% de complejidad cognitiva y un 18% de avisos de análisis estático, y que esa deuda se acumula de forma persistente en vez de autocorregirse.

La disciplina se paga sola. Los equipos que tratan los tokens como recurso finito, aplican context engineering, enrutan al modelo más pequeño que pasa el eval e instrumentan el costo pagan entre un 30% y un 70% menos que los equipos que usan GitHub Copilot como caja negra. En cargas agénticas en concreto, los patrones de optimización aplicados se acumulan hasta una reducción en el mismo rango. La diferencia entre el cementerio y la plataforma no es talento. Es disciplina aplicada, en secuencia, a lo largo de seis capas.

DEFINICIÓN

Todo el playbook en una frase: los pilotos mueren por tres deudas que se acumulan, seis disciplinas de ingeniería en secuencia mantienen a los agentes vivos en producción, y desde el 1 de junio de 2026, cuando GitHub AI Credits hizo el costo por trabajo realizado, esa disciplina paga entre un 30% y un 70% menos en la factura.

Referencias

Fuentes primarias para el patrón de falla, la pila de seis disciplinas y los dos hechos que justifican la plataforma.

- Deloitte, State of AI in the Enterprise 2026, la encuesta detrás de la brecha: el 75% de las empresas planea IA agéntica, solo el 34% reporta transformación profunda.
- Debt Behind the AI Boom: A Large-Scale Empirical Study of AI-Generated Code in the Wild, la evidencia de deuda técnica en 304.362 commits generados por IA en 6.275 repositorios.
- AI IDEs or Autonomous Agents? Measuring the Impact of Coding Agents on Software Development, la medición causal de cómo los agentes de codificación cambian la salida de ingeniería.
- GitHub Docs, Usage-based billing for GitHub Copilot: GitHub AI Credits, el cambio de facturación de junio de 2026: 1 AI Credit equivale a US$0,01, costo por trabajo realizado.
- CNCF, Platforms Whitepaper, la referencia para la disciplina de plataforma y los golden paths.
- Spec Kit, toolkit for Spec-Driven Development, la herramienta detrás de la disciplina de intención: constitución, especificación y plan de implementación.
- Anthropic, Effective context engineering for AI agents, guía fundamental para la disciplina de contexto.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

01 SUSTRATO Y GOBERNANZA

Fundación cloud e ingeniería de plataforma.

Las dos primeras capas de la pila deciden si un agente llega a producción. La infraestructura cloud da a las cargas agénticas un lugar donde ejecutarse. La ingeniería de plataforma decide qué tienen permitido hacer esas cargas. Este capítulo cubre ambas: cómo cambia Kubernetes cuando la carga es un agente de larga vida en vez de un request sin estado, por qué el Model Context Protocol reduce el costo de integración, cómo los golden paths y la identidad de workload convierten la gobernanza de documento en código aplicado, y cómo la arquitectura CPE de cuatro planos y los controles de cadena de suministro sostienen la pila agéntica.

Kubernetes para cargas agénticas

Kubernetes se ha convertido en el runtime por defecto para IA en producción. Dos de cada tres organizaciones que ejecutan IA a escala lo usan como plataforma primaria de orquestación. No ganó por haber sido hecho para machine learning. Ganó porque ya resolvía cargas distribuidas y con estado a escala enterprise. La Cloud Native Computing Foundation entonces extendió esa base para IA. En marzo de 2026, el Kubernetes AI Conformance Program certificó más de un 70% de plataformas adicionales, una señal de que Kubernetes estándar necesita extensiones deliberadas para ejecutar agentes con seguridad.

Lo que cambia es el perfil de la carga. Un job de entrenamiento se ejecuta hasta terminar. Un servicio de inferencia responde a un request y retorna. Un agente no hace ninguna de las dos cosas. Las sesiones agénticas viven durante horas, mantienen KV cache entre pasos y hacen fan-out a subagentes. Tres capacidades de infraestructura atacan esto de forma directa. El redimensionamiento de pod in-place deja que un agente pase de planificación ligera a uso intensivo de tools sin reiniciar, así el pod sigue agendado y solo cambia su reserva. El scheduling consciente de la carga entiende la coordinación entre agentes y rompe las esperas

circulares que atrapan al scheduling FIFO estándar. El sandboxing da a cada agente aislamiento a nivel de sistema operativo o de VM, una frontera que se sostiene incluso cuando el agente ejecuta código que no escribió. Alrededor del cómputo están los requisitos operativos: contenedores de menor privilegio, secretos con alcance y la pila de observabilidad MELT de métricas, eventos, logs y traces, para que las decisiones que un agente toma sigan siendo auditables en vez de opacas.

MCP como la capa de protocolo

Sin un protocolo común, cada agente integra cada herramienta en sus propios términos. Eso son N por M integraciones punto a punto, y la cuenta crece con los agentes y las herramientas a la vez. El Model Context Protocol lo reduce a N más M. Cada agente habla MCP una vez, cada herramienta expone un servidor MCP una vez, y cualquier agente alcanza cualquier servidor. El MCP lleva tres primitivas: tools que producen efectos secundarios, resources que exponen datos y prompts que moldean la interacción. GitHub Copilot en agent mode consume servidores MCP de la misma manera, lo que significa que la integración que construyes para la plataforma es la integración que usa Copilot.

El valor arquitectónico es la frontera, no el formato del cable. Un agente no toca una base de datos ni una API externa directamente. Habla MCP con un servidor que media el acceso, y un gateway MCP propiedad de la plataforma centraliza autenticación, alcance y telemetría delante de esos servidores. Política, rate limits y clasificación de contenido viven en el gateway, aplicados sin cambiar el código del agente. El MCP es deliberadamente estrecho. Conecta agentes con datos, herramientas y servicios, y no intenta ser dueño de la observabilidad, la identidad de workload ni el control de acceso. Esa contención es lo que lo mantiene combinable con OpenTelemetry para traces y con SPIFFE para identidad. A principios de 2026, el protocolo gobernaba más de 10.000 servidores activos, con 97 millones de descargas mensuales de SDK, y pasó a la Agentic AI Foundation en la Linux Foundation, que superó las 170 organizaciones miembro para marzo de 2026.

Golden paths e identidad

La ingeniería de plataforma ya es obligatoria. La evaluación DORA 2025 reportó que cerca del 90% de las empresas ya operan una plataforma interna, y Backstage, la plataforma de desarrollo interna open-source, tiene cerca del 89% de participación entre adoptantes. La razón es la gobernanza. Sin golden paths, cada equipo reconstruye los mismos controles, y cada agente se entrega con su propia autenticación, su propia telemetría y su propio manejo de secretos. Con golden paths entregados como scaffolds de Backstage, templates de IDP y registros de servidor MCP, los controles de seguridad y de costo vienen compilados. Los equipos son conformes por defecto y solo optan por salir con una razón declarada.

Un archivo .github/copilot-instructions.md es un golden path en el mismo sentido. Codifica las convenciones que Copilot debe seguir para que el agente empiece dentro de las guardas en vez de adivinar. Los cuatro pilares del control de plataforma, golden paths, guardrails, redes de seguridad y compuertas de revisión manual, aplican a los agentes exactamente como aplican a las personas. Los agentes se vuelven ciudadanos de primera clase de la plataforma, con su propio RBAC, sus propias cuotas de recursos y sus propios presupuestos de costo.

Identidad y política como código

La identidad es la otra fundación. Los agentes necesitan identidades de workload SPIFFE, no service accounts compartidas. Cada agente lleva un SPIFFE ID que se propaga del llamador al servidor MCP a la fuente de datos, y los tokens se acuñan por sesión con alcance estrecho y TTL corto, para que un agente nunca exceda lo que el usuario solicitante podría hacer. La política como código cierra el ciclo. Las reglas escritas para OPA o Cedar viven en repositorios, versionan junto al código que gobiernan y pasan revisión de pares como cualquier otro artefacto de infraestructura. El log de auditoría que un revisor necesita es consultable sin ayuda de ingeniería.

La arquitectura de cuatro planos y la seguridad de la cadena de suministro

El Cognitive Platform Engineering da a la pila una forma de referencia. Organiza la plataforma en cuatro planos construidos sobre Kubernetes, Terraform y OPA. Un data plane ejecuta cargas de agente y servidores MCP en pods con GPU. Un control plane maneja la orquestación mediante la API de Kubernetes, GitOps y Argo. Un cognition plane lleva enrutamiento, política y artefactos de intención como especificaciones de Spec Kit y la CONSTITUTION del proyecto. Una capa de observabilidad recopila spans, logs y datos de costo mediante OpenTelemetry.

Los planos cierran un ciclo sentir-razonar-actuar. La plataforma siente patrones de carga, razona sobre la configuración y actúa mediante el control plane, para que un agente que pasa de planificación a generación de código encuentre recursos antes de que un desarrollador abra un ticket.

Asegurando la cadena de suministro agéntica

La superficie de seguridad es más amplia que la seguridad de aplicación. Un análisis de 295 GitHub Security Advisories que referencian componentes de LLM encontró vulnerabilidades de cadena de suministro en el 44% de ellos, por delante de la agencia excesiva con 20% y la inyección de prompt con 18%. Los controles son conocidos y baratos. Combina al menos dos scanners de GitHub Actions, ya que ningún scanner por sí solo cubre toda clase de debilidad, y el escaneo mediano cuesta 2,71 segundos por workflow, así que no hay excusa de rendimiento para omitirlo. Genera procedencia SLSA y un SBOM en el pipeline, firma los artefactos y trata todo cambio generado por agente como no confiable hasta que la política lo libere. La facturación refuerza la disciplina. Desde el 1 de junio de 2026, GitHub Copilot cobra en GitHub AI Credits a un crédito por US$0,01, medido sobre los tokens de input, output y cached que cada modelo consume, así que un agente derrochador aparece como línea de costo y uno bien acotado no.

DEFINICIÓN

Las capas uno y dos en una frase: Kubernetes da a los agentes un runtime hecho para sesiones de larga vida y con estado, y la ingeniería de plataforma da a ese runtime sus reglas. Sustrato antes que autonomía, protocolo en vez de punto a punto, identidad por sesión, política como código y verificaciones de cadena de suministro en cada pipeline.

Referencias

Fuentes primarias para la fundación cloud, la capa de protocolo, los golden paths y la seguridad de la cadena de suministro agéntica.

- CNCF, Platforms Whitepaper, la definición de referencia de plataformas internas y golden paths.
- Cognitive Platform Engineering: 4-Plane Reference Architecture, el modelo de cuatro planos sobre Kubernetes, Terraform y OPA.
- Model Context Protocol: Design Patterns and Security Considerations, estructura del protocolo y el patrón de gateway propiedad de la plataforma.
- LLM-Enabled OSS Vulnerabilities in GitHub Security Advisories, la porción de cadena de suministro en los advisories de la pila agéntica, 44% de 295.
- Unpacking Security Scanners for GitHub Actions, por qué dos scanners superan a uno, a 2,71 segundos de mediana por workflow.
- GitHub Docs, Usage-based billing for GitHub Copilot: GitHub AI Credits, cómo se mide el consumo de tokens desde el 1 de junio de 2026.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

02 LA CAPA DE COGNICIÓN

Ingeniería de contexto: lo que el agente sabe.

La ingeniería de contexto es la disciplina de ensamblar el conjunto más pequeño de tokens de alta señal que maximiza la calidad de la salida de un agente. No es llenar la ventana con todo lo que podría ayudar. Es lo contrario: curaduría hecha con bisturí. En GitHub Copilot en agent mode, el contexto que el agente lee decide tanto la respuesta como la cuenta. Este capítulo cubre la curva de utilidad, los niveles de memoria, seis técnicas que se componen y el prompt caching.

La curva de utilidad cóncava: más contexto no es mejor

La intuición común es que más contexto siempre ayuda al modelo. Más información, mejor respuesta. Esa intuición es incorrecta. La relación entre la cantidad de contexto y la calidad de la salida es cóncava con pico, no creciente de forma monotónica. Agregar tokens irrelevantes tiene un efecto neutro al principio y negativo después. Esto es context rot. Tres mecanismos lo provocan. La dilución de señal viene primero: para una pregunta sobre un módulo, inyectar cincuenta archivos de otro módulo obliga al modelo a filtrar ruido antes de poder enfocarse. La ventana de atención es finita: incluso modelos con ventana de un millón de tokens aplican un gradiente de atención, así que los tokens del medio reciben menos atención que los del principio y el final, y la dilución empuja el material relevante a la zona de baja atención. El costo sube sin beneficio: más tokens siempre cuestan más dinero, y si la calidad no aumenta, el gasto es puro desperdicio.

La regla a aplicar antes de agregar cualquier cosa

Anthropic enuncia la regla sin rodeos: usa el conjunto más pequeño de tokens de alta señal que maximiza la probabilidad del resultado deseado. No todo el contexto posible. No el contexto que pareció relevante a primera vista. El mínimo que sostiene el resultado correcto. Antes de inyectar contexto, haz tres preguntas. ¿Esta

información es necesaria para esta tarea específica? ¿Puede recuperarse bajo demanda en vez de inyectarse de antemano? ¿La agrego porque ayuda, o porque se siente como una garantía? La respuesta de garantía suele ser la señal de que se está introduciendo context rot.

Niveles de memoria y MCP: guías que el agente lee primero

El contexto vive en dos niveles, y confundirlos es caro. La ventana de contexto es la memoria de trabajo de una sola petición. Es volátil, costosa y finita, de doscientos mil a un millón de tokens según el modelo, y cada turno relee todo. La memoria persistente es información almacenada fuera del contexto, en un vector store, un key-value store o una base de datos, y recuperada bajo demanda. Acumula indefinidamente sin inflar la ventana, el almacenamiento cuesta centavos por gigabyte, y la recuperación solo cuesta cuando se consulta. La pregunta de prueba decide dónde pertenece un hecho: ¿esto está en uso activo en esta interacción, o es referencia potencialmente útil? El uso activo va a la ventana de contexto. La referencia potencialmente útil va a la memoria persistente. La recuperación ocurre por el Model Context Protocol (MCP), la interfaz estándar que el agente usa para alcanzar un store, una tool o un resource sin una integración a medida para cada uno.

Los archivos de instrucciones son contexto curado por humanos

La forma concreta de contexto curado en el flujo del desarrollador es el archivo de instrucciones. El AGENTS.md en la raíz del repositorio y el .github/copilot-instructions.md se leen automáticamente al inicio de una sesión, antes de cualquier tarea. Un estudio sobre AGENTS.md midió el efecto: un archivo curado por humanos reduce el runtime en 28,64% y el uso de tokens en 16,58%. Un archivo generado por LLM rinde 3% peor que ningún archivo. La curaduría humana es obligatoria. El archivo .github/copilot-instructions.md es la forma de GitHub Copilot, leída por Copilot Chat y por el Copilot coding agent y renderizada en la interfaz de VS Code. Mantén estos archivos ligeros, de cincuenta a doscientas líneas, y actualízalos en pull requests cuando cambien las convenciones. Codifican dónde vive cada cosa, las convenciones que sigue la base de código y lo que el agente nunca debe tocar.

Seis técnicas que se componen y recortan el costo recurrente de tokens

Seis técnicas se componen sobre la curaduría, cada una medida. La compaction es el modelo comprimiendo su propia historia de sesión en un resumen estructurado que conserva decisiones, archivos modificados y errores, y descarta intentos fallidos y relleno. Medida en un benchmark de código, entregó 22,7% de reducción media de tokens, hasta 57% en sesiones muy largas, con 0% de pérdida de exactitud. El tool-result clearing ataca al mayor consumidor de tokens en sesiones largas: la lectura de un solo archivo puede inyectar treinta mil tokens, y los resultados de tool antiguos permanecen en el contexto y se pagan en cada llamada posterior si no se limpian. Conserva los últimos cinco a diez resultados y reemplaza el resto por un marcador corto.

Los subagentes E-mem aíslan el contexto por fan-out: un lead despacha subagentes que exploran cada uno una porción y devuelven solo un resumen, lo que alcanzó la misma calidad de tarea con 70% menos tokens. La memoria persistente mueve hechos estables fuera de la ventana; un estudio destiló 4.182 conversaciones en una capa compacta con 11x de compresión manteniendo la recuperación intacta. El SkillReducer adelgaza archivos de skill de forma programática, 39% de reducción en el cuerpo y 48% de reducción en la descripción. La sexta técnica, prompt caching, es la mayor palanca individual y tiene su propia sección. Estas técnicas se apilan: la compaction encoge la historia, el clearing elimina resultados obsoletos, los subagentes aíslan la exploración, la memoria persistente descarga la referencia, y el SkillReducer afina las definiciones que el agente siempre lee.

DEFINICIÓN

Seis técnicas que se componen: compaction (22,7% de reducción media de tokens, 0% de pérdida de exactitud), tool-result clearing (conserva los últimos cinco a diez resultados), subagentes E-mem (70% menos tokens), memoria persistente (11x de compresión en la recuperación), SkillReducer (39% de cuerpo más pequeño, 48% de descripción más pequeña) y prompt caching (hasta 90% de ahorro de input). Cura el conjunto más pequeño de alta señal y luego reutilízalo.

Prompt caching: reutilizar la KV cache de un prefijo estable

Cada llamada al modelo ejecuta el mecanismo de atención sobre todos los tokens, el trabajo más caro de la petición, y almacena el resultado intermedio, la KV cache de atención, en la memoria de la GPU. En una petición posterior con el mismo prefijo de tokens, el proveedor recupera esa KV cache en vez de recomputarla. El modelo se salta el trabajo de procesar esos tokens de nuevo. En cargas conversacionales, esto reduce el costo de input hasta en 90% sin cambiar una sola línea de prompt. La cache opera por hash de prefijo, byte a byte, así que cualquier cambio en cualquier byte invalida la cache desde ese punto en adelante.

La disciplina es una regla: contenido estable primero, contenido dinámico después. Pon el system prompt, las definiciones de tool y el contexto estático al inicio y márcalos como cacheables; mantén el mensaje del usuario y todo lo que cambia al final. Un system prompt modificado invalida el prefijo entero, así que versiónalo con cuidado. Bajo los GitHub AI Credits, los tokens cacheados se cobran a la tarifa publicada de cada modelo, donde un crédito equivale a un centavo, así que un prefijo estable no solo es más rápido, es un renglón de costo ahorrado en cada llamada repetida. Al enrutar por el selector de Copilot, la misma disciplina de prefijo aplica, ya sea que la llamada vaya al Haiku 4.5 en un paso trivial, al Sonnet 5 para código, o al Opus 4.8 para un plan.

Referencias

Fuentes primarias para la curva de utilidad, la evidencia de los archivos de instrucciones, el estándar de recuperación y el modelo de facturación.

- Anthropic, Effective context engineering for AI agents, la fuente de la regla: el conjunto más pequeño de tokens de alta señal que maximiza la probabilidad del resultado deseado.
- GitHub Docs, Customizing GitHub Copilot with instruction files, cómo se lee y aplica el .github/copilot-instructions.md en agent mode.
- Spec Kit, toolkit for Spec-Driven Development, el patrón de artefactos de especificación curados por humanos que anclan agentes personalizados.

- GitHub Docs, Usage-based billing and GitHub AI Credits, cómo se miden los tokens cacheados y no cacheados, donde un AI Credit equivale a un centavo.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

03 LA CAPA ESTRATÉGICA

Intent engineering: lo que el agente debe hacer.

El context engineering hace preciso a un agente. No hace que el agente apunte al objetivo correcto. Esa es la brecha que cierra el intent engineering. Un agente de GitHub Copilot en agent mode puede cargar el repositorio entero en contexto y aun así optimizar para un resultado que la organización nunca eligió. Este capítulo es la capa estratégica de la pila. Muestra cómo codificar lo que el agente debe hacer en artefactos versionados: CONSTITUTION.md para valores, SPECIFICATION.md para requisitos testeables e IMPLEMENTATION_PLAN.md para alcance. Luego muestra por qué la interfaz entre planificación y codificación es donde comienza la mayoría de las fallas multi-agente.

Deuda de intención: contexto completo, sin objetivo, resultado equivocado

Vuelvo siempre al caso Klarna del tercer trimestre de 2025 porque muestra exactamente qué se rompe. Klarna desplegó un agente con todo lo que se quiere desde la infraestructura: datos detallados de clientes, especificaciones de producto, reglas de precios, historial de ejecución. El contexto era exhaustivo. En semanas, el agente empezó a optimizar para métricas que la organización nunca quiso priorizar. Cuando alguien notó el desalineamiento, el sistema ya había incurrido en costo operativo y reputacional. La lección es directa: el context engineering no impide que un agente optimice para lo equivocado. Puedes darle todo el contexto del mundo a un agente, pero si nunca le dices qué optimizar, optimizará por algo.

La deuda de intención se acumula en silencio. Los objetivos, las restricciones y el razonamiento detrás de ellos viven en un hilo de Slack de hace ocho meses, en un correo perdido en un canal de soporte, en la cabeza de un solo ingeniero. Cada uno es un punto de falla en un sistema pensado para operar de forma autónoma a escala. Cada generación de trabajo asistido por IA sin artefactos explícitos de

intención ensancha la distancia entre lo que el sistema debía hacer y lo que hace. El artículo del tema nombra tres deudas que se acumulan juntas: técnica, cognitiva y de intención. La deuda de intención es la más insidiosa porque el agente no está roto. Simplemente nunca se especificó por completo.

> DEFINICIÓN
>
> Un agente bien contextualizado sin intención escrita es una flecha con excelente puntería y sin objetivo. El contexto le dice al agente cómo funciona el mundo. La intención le dice a qué debe apuntar.

Los artefactos del Spec Kit: valores, requisitos, alcance

El intent engineering se materializa en tres artefactos, y el Spec Kit los hace legibles. CONSTITUTION.md es un documento human-first de principios, guardrails y valores organizacionales no negociables. Se escribe para que lo lean humanos y se estructura para que lo apliquen máquinas. Una constitución declara una jerarquía de lo que más importa cuando surgen trade-offs, por ejemplo, que la optimización de costo nunca tiene precedencia sobre la privacidad de datos. No es un checklist de cada decisión. En un repositorio con GitHub Copilot, esos mismos principios llegan al agent mode mediante el archivo .github/copilot-instructions.md, de modo que los valores viajan junto con el código.

SPECIFICATION.md codifica requisitos en un formato testeable y legible por máquina. Traduce un principio constitucional en una condición que el agente puede verificar antes de actuar. Si la constitución dice preservar la transparencia, la especificación dice: cuando se descubra una limitación del sistema, escala a un revisor humano antes de continuar. IMPLEMENTATION_PLAN.md descompone el trabajo en tareas atómicas y marca cuáles pueden correr en paralelo. El plan se convierte en una frontera de control. El agente sabe exactamente qué puede modificar, y todo lo que está fuera del plan queda fuera de alcance por defecto.

DEFINICIÓN

Tres archivos, tres funciones. CONSTITUTION.md fija los valores. SPECIFICATION.md fija los requisitos testeables. IMPLEMENTATION_PLAN.md fija el alcance. Juntos son la columna vertebral del Spec-Driven Development.

Spec-Driven Development: la compuerta, los hooks, la caída de defectos

El Spec-Driven Development organiza el trabajo del agente en fases, con un control rígido entre ellas. Primero, un humano y un planner redactan CONSTITUTION.md y SPECIFICATION.md en Ask mode, sin escritura de archivos. Luego refinan la especificación en un IMPLEMENTATION_PLAN.md. Entonces llega la compuerta de aprobación humana. El agente no entra en agent mode, donde tiene permiso de escritura, hasta que un humano aprueba el plan. No es una frontera blanda. La impone la capa de orquestación. La compuerta mueve el juicio humano hacia arriba, dentro de la especificación, donde un desalineamiento todavía es barato de corregir.

Hooks: enforcement a cero AI Credits

Los hooks aplican el plan a costo cero de tokens. Un hook preToolUse es un comando de shell que intercepta una escritura y la bloquea cuando el archivo destino no está en IMPLEMENTATION_PLAN.md. Corre en la capa de orquestación, no en el modelo, así que no consume ningún GitHub AI Credit. El agente propone, el hook aplica, el humano decide si modifica el plan. La investigación sobre SDD constitucional encontró que reduce los defectos de seguridad en 73% frente al desarrollo asistido por IA sin restricciones. El mecanismo no es una especificación perfecta. Es que la revisión explícita del plan antes de la implementación expone el desalineamiento mientras todavía se puede corregir. Enruta la fase de planificación a un modelo más fuerte en el Copilot picker, como Opus 4.8 o Claude Fable 5, y la implementación con alcance acotado a Sonnet 5.

DEFINICIÓN

La compuerta humana entre especificación e implementación es el control más importante del flujo. Los hooks la hacen barata de mantener: un condicional de shell en la capa de orquestación, cero tokens de modelo, cero AI Credits.

La brecha planner-coder: la especificación es la interfaz

Un estudio de fallas de sistemas multi-agente encontró que el 75,3% de ellas se origina en la brecha entre el planner y el coder. Cuando el planner escribe una especificación ambigua o incompleta, el coder genera código inseguro o incorrecto. Esto no es solo un problema de calidad. Es un vector de seguridad. Un atacante que logre influir en lo que el planner produce, mediante prompt injection o manipulación de contexto, puede hacer que el coder genere código vulnerable sin tocar nunca al coder.

Este es el argumento empírico más fuerte para tratar SPECIFICATION.md como ingeniería de verdad, no como papeleo. La especificación es la interfaz entre planificación y codificación. Cuando esa interfaz es explícita, legible por máquina y revisada por humanos, cierras la brecha donde comienzan tres cuartos de las fallas. Cuando es implícita, una tarea vaga o un prompt casual, dejas esa superficie de falla abierta. La misma base de evidencia muestra que el modelo es un motor de resultado mayor que el framework, así que invierte en la especificación y en la elección de modelo en el Copilot picker antes de invertir en la elección de framework. El intent engineering es la capa estratégica. El harness engineering, en el próximo capítulo, es el modelo operativo que lo ejecuta.

DEFINICIÓN

El 75,3% de las fallas de los sistemas multi-agente comienza en la interfaz planner-coder. La especificación es esa interfaz. Hazla explícita, legible por máquina y revisada por humanos, y cierras tres cuartos de la superficie de falla.

Referencias

Fuentes primarias para la deuda de intención, los artefactos del Spec Kit, el Spec-Driven Development y la evidencia empírica sobre costo, complejidad y la interfaz de especificación.

- GitHub, Spec Kit: a toolkit for Spec-Driven Development, el toolkit que convierte CONSTITUTION.md, SPECIFICATION.md e IMPLEMENTATION_PLAN.md en artefactos concretos y versionados.
- GitHub Docs, Usage-based billing for individuals: GitHub AI Credits, cómo se mide el trabajo del agente en GitHub AI Credits, un crédito equivalente a un centavo de dólar desde el 1 de junio de 2026, y por qué los hooks en la capa de orquestación consumen cero créditos.
- arXiv, Debt Behind the AI Boom: A Large-Scale Empirical Study of AI-Generated Code in the Wild, evidencia a gran escala de que el código generado por IA desplaza la deuda hacia requisitos y pruebas, la materia prima de la deuda de intención.
- arXiv, AI IDEs or Autonomous Agents? Measuring the Impact of Coding Agents on Software Development, el impacto medido de los coding agents, incluida la complejidad cognitiva que los agentes sin gobernanza añaden a las organizaciones de ingeniería.
- Deloitte, State of AI in the Enterprise 2026, la brecha de transformación que la deuda de intención ensancha: tres de cada cuatro empresas planean IA agéntica, cerca de un tercio reporta transformación profunda.
- Anthropic, Effective context engineering for AI agents, la disciplina sobre la que se apoya el intent engineering, el contexto hace preciso al agente, la intención le dice a qué apuntar.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

04 EJECUCIÓN Y OPERACIONES

Harness Engineering: el modelo operativo.

Un agente es un modelo más un harness. El modelo razona. El harness es todo lo que lo rodea: tools, memoria, estado, gates, sensores y guías. Cambia el modelo en el picker de GitHub Copilot y el agente sigue comportándose como su harness lo permite. Este capítulo convierte el stack en un modelo operativo: la taxonomía de guías y sensores, cuatro patrones de producción, siete pasos de implementación y dos aceleradores sobre una fundación Microsoft.

Agente es igual a modelo más harness

La fórmula que nombró el campo a principios de 2026 es compacta: un agente es un modelo más un harness. El modelo aporta la capacidad de razonamiento. El harness es todo lo demás, las tools, la memoria, el estado, el contexto, la validación, la seguridad, el ciclo de vida, los gates, los sensores y las guías. En GitHub Copilot el modelo es la parte intercambiable. Lo eliges en el picker, Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro o GPT-5.x, y el harness permanece constante. Beren Millidge lo planteó en 2023: un modelo crudo es una CPU sin sistema operativo, y el harness es el SO que la hace útil.

Martin Fowler y Birgitta Böckeler dieron al harness su taxonomía más clara en marzo de 2026. Un harness se divide en guías, que orientan al agente, y sensores, que lo verifican. Las guías son los archivos que el agente lee primero: .github/copilot-instructions.md, AGENTS.md, system prompts versionados, documentos de restricción arquitectónica y descripciones ricas de tools. Los sensores observan lo que el agente hizo. Los sensores deterministicos son baratos y rápidos: linters personalizados, type checkers, pre-commit hooks, tests estructurales y gates de CI que bloquean el merge cuando cualquier verificación falla. Los sensores inferenciales se basan en LLM: code review como juez y análisis semántico de pull request. Un estudio de archivos AGENTS.md encontró que los archivos curados por humanos redujeron el runtime en 28,64% y el uso de tokens en 16,58%, mientras

que los archivos generados por LLM rindieron 3% peor que ningún archivo. La disciplina es la curaduría, no el tamaño del archivo.

> DEFINICIÓN
>
> Agente es igual a modelo más harness. El modelo es el componente intercambiable que seleccionas en el picker de GitHub Copilot. El harness es lo que tu equipo construye y mantiene: guías que orientan al agente y sensores que lo verifican. Construye un agente y lo que construiste fue un harness apuntado a un modelo.

Patrones de producción

Cuatro patrones se repiten en producción. El ReAct loop es el más simple: el agente razona, elige una tool, observa el resultado e itera hasta alcanzar el objetivo o agotar su presupuesto. Úsalo para tareas cortas con una verificación de éxito clara: triage de issues, resumen, generación de reportes. El patrón initializer más coding agent maneja trabajo que se extiende por muchas sesiones. Un initializer corre una vez y crea la estructura: un init script, un archivo de progreso como log de sesión a sesión, un primer commit y una feature list de ítems rastreables. Un coding agent luego corre en cada sesión posterior: lee el progreso, elige la siguiente feature, implementa, prueba con Playwright, actualiza el log y hace el commit. El estado vive en archivos versionados, así la memoria de largo plazo queda legible y auditable.

El patrón Planner-Generator-Evaluator, de Anthropic en marzo de 2026, divide el trabajo en tres roles. El Planner expande un prompt corto en una especificación y deja deliberadamente los detalles de implementación abiertos. El Generator implementa en sprints y antes firma un sprint contract con el Evaluator: una definición de terminado compartida. El Evaluator conduce la aplicación con Playwright como un usuario real y falla el sprint si algo se rompe. La investigación encontró que el 75,3% de las fallas de multi-agente se originan en la interfaz entre planner y coder, y el Evaluator la hace explícita. El cuarto patrón es la garbage collection continua. Agentes de fondo recorren el repositorio en cadencia buscando imports que cruzan capas, duplicación y deriva, luego abren pequeños pull requests de refactorización que se revisan en menos de un minuto y hacen auto-merge.

OpenAI trata la deuda técnica como un préstamo de interés alto, pagada continuamente en vez de en brotes dolorosos.

El playbook de siete pasos

Los siete pasos son secuenciales y acumulativos. Primero, elige el caso de uso correcto. Uno bueno tiene un criterio de éxito verificable, trabajo repetitivo que justifica la inversión, impacto material y riesgo contenible. Criterios vagos, trabajo puntual y riesgo ilimitado son donde mueren los pilotos. Segundo, escribe el archivo de instrucción como un mapa, no una enciclopedia. Para GitHub Copilot eso es el .github/copilot-instructions.md. Mantenlo cerca de 100 líneas y deja que apunte a fuentes más profundas en un directorio docs estructurado. Tercero, instrumenta trayectorias desde el día uno: el registro completo de una sesión, prompts, llamadas de tool, salidas, errores y tiempos. Sin ella no depuras, y cada sesión capturada se vuelve dato de evaluación. Cuarto, planta sensores determinísticos antes de cualquier juez LLM. Un linter personalizado para reglas arquitectónicas, un type checker en cada pull request, pre-commit hooks y un gate de CI obligatorio atrapan la mayoría de las fallas de forma barata.

Quinto, impón la arquitectura mecánicamente. Codifica la dirección de dependencia como un linter en vez de un párrafo. En el stack Microsoft, combina Bicep con Azure Policy, analizadores Roslyn y branch protection en GitHub. Este es el paso que separa los pilotos de la producción. Sexto, agrega loops de autoverificación: sprint contracts, orquestación test-first, automatización de navegador Playwright y loop detection middleware que aborta en acciones repetidas. LangChain saltó del rank 30 al rank 5 en Terminal Bench 2.0 agregando sensores y autoverificación con el modelo mantenido constante. La pieza faltante suele ser un sensor, no un modelo mejor. Séptimo, ejecuta el loop de mejora continua. Observa una falla en una trayectoria, clasifica la causa raíz, haz que el agente escriba la corrección bajo revisión, agrega un test de regresión y audita trayectorias futuras. Cada falla se vuelve estructuralmente imposible de repetir. Tres ingenieros usando esta disciplina entregaron cerca de un millón de líneas de código de producción sin código escrito por humanos.

Dos caminos sobre una fundación Microsoft

La fundación bajo cada compromiso es constante. Azure es la cloud, GitHub Enterprise lleva el control de fuente y el DevOps, Entra Agent ID da a cada agente una identidad gestionada con alcances y traza de auditoría, y Azure AI Foundry con Microsoft Agent Framework es la plataforma de IA. GitHub Copilot entrega las primitivas del harness directamente. El archivo .github/copilot-instructions.md es la guía del repositorio; archivos .instructions.md con alcance por ruta y archivos .prompt.md reutilizables lo extienden. Los custom chat modes guardan las personas del equipo, los agentic workflows llevan loops y gates, y la configuración de servidor MCP expone las tools. El GitHub Copilot coding agent maneja el trabajo multi-sesión, y la automatización de code review actúa como sensor inferencial en los pull requests. El cobro corre en GitHub AI Credits desde el 1 de junio de 2026: un crédito equivale a un centavo de dólar, y los tokens de input, output y cached se miden a la tarifa publicada de cada modelo, así el costo sigue el trabajo realizado, no los asientos.

Dos aceleradores materializan el mismo harness. Three Horizons usa Red Hat Developer Hub sobre Azure Red Hat OpenShift para clientes con inversión previa en Red Hat o requisitos fuertes de soporte. Open Horizons usa Backstage sobre Azure Kubernetes Service para clientes Azure-native. Ambos implementan la misma disciplina y llegan al mismo destino Horizon 3; solo cambian la plataforma interna de desarrollo y el cluster. El mapeo de modo de falla a capa de harness vuelve mecánica la revisión de incidentes: los imports que cruzan capas apuntan a un linter de dirección de dependencia, la conclusión prematura a un sprint contract y un evaluator separado, un agent loop a loop detection middleware, un secreto filtrado a un pre-commit hook y un sandbox sin acceso a secretos, y un comando destructivo a una política de aprobación obligatoria. Ninguno se resuelve con un prompt mejor. Cada uno necesita una pieza estructural del harness, y las primitivas de agent mode en GitHub Copilot proveen la mayoría de fábrica.

Referencias

Fuentes primarias para la fórmula del harness, la taxonomía de guías y sensores, los cuatro patrones de producción y el modelo operativo de siete pasos.

- Mitchell Hashimoto, My AI Adoption Journey, origen de la disciplina: ingeniar el harness para que el agente nunca repita un error.
- Ryan Lopopolo, OpenAI, Harness engineering in an agent-first world, el playbook de siete pasos, la garbage collection continua y cerca de un millón de líneas de código sin código escrito por humanos.
- LangChain, Improving Deep Agents with harness engineering, del rank 30 al rank 5 en Terminal Bench 2.0 cambiando el harness con el modelo mantenido constante.
- Birgitta Böckeler, Harness engineering for coding agent users, martinfowler.com, la taxonomía de guías y sensores sobre la que se apoya este capítulo.
- Anthropic, Harness design for long-running application development, el patrón Planner, Generator y Evaluator para sesiones autónomas de varias horas.
- Microsoft Agent Framework 1.0, la capa gestionada Agent Harness sobre la fundación compartida Azure y GitHub.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

05 FINANCIAMIENTO Y CIERRE

Economía de tokens, FinOps y adopción: financia la plataforma, jubila el cementerio.

La plataforma está construida. La factura sigue llegando. El 1 de junio de 2026, GitHub Copilot dejó de cobrar por request y pasó a cobrar por los tokens que cada modelo realmente consume, así que la elección de modelo, el caché y el enrutamiento se vuelven líneas que un equipo de finanzas puede leer. Este capítulo financia la plataforma: cómo enrutar modelos en el selector de Copilot, cómo operar FinOps para IA, cómo recortar del 25 al 45 por ciento sobre el baseline y cómo adoptar a escala enterprise.

Enrutamiento de modelos en el selector de Copilot: paga cada modelo a su tarifa de API

El selector de Copilot expone un modelo por interacción, y el agent mode lee esa elección en cada tarea. Enruta por complejidad. Haiku 4.5 se encarga de las tareas triviales: clasificación, extracción y trabajo en lote de baja matización. Sonnet 5 es el caballo de batalla para código: debugging, refactorizaciones multiarchivo y revisión profunda. Opus 4.8 queda reservado para planes: arquitectura y decisiones que se reutilizarán. Cada llamada premium contabiliza tokens de input, output y cached a la tarifa de API publicada de ese modelo, convertida a 1 AI Credit igual a US$0,01. Los modelos bundled incluidos en el plan no gastan créditos.

El enrutamiento es la palanca de costo de primer orden, y su lugar es el control de versiones, no la cabeza de cada desarrollador. Fija el valor por defecto en Sonnet 5 para agent mode y edit mode, ancla Opus 4.8 al plan mode y baja a Haiku 4.5 para tareas en lote bien definidas. Haz commit de esa política en .vscode/settings.json y declara la justificación en .github/copilot-instructions.md para que todo el equipo ejecute el mismo enrutamiento. El tier premium cuesta varias veces el del caballo de

batalla para la misma sesión, así que un valor por defecto de Opus 4.8 es dinero gastado sin una ganancia equivalente en calidad.

FinOps para IA: Inform, Optimize, Operate

FinOps para IA es la disciplina de instrumentar, medir, optimizar y operar el gasto de tokens con el rigor aplicado a cualquier otro recurso de nube. Corre en tres fases. Inform captura telemetría por request, modelo, tokens, usuario y feature, mediante OpenTelemetry, la exporta bajo el schema FOCUS y la muestra en dashboards de Power BI o Grafana. Sin datos, la optimización es adivinar. Empieza por el showback: muestra a cada equipo el costo que generó sin cobrarle, porque el comportamiento cambia cuando el número se vuelve visible.

Optimize identifica el desperdicio específico: prompts ineficientes, modelos sobredimensionados, falta de caché, equipos sin entrenar. Cada corrección tiene un payback estimable, medible y atribuible. Operate cierra el ciclo con chargeback a los equipos, detección de anomalías que marca una sesión corriendo tres sigma sobre la media y revisión trimestral de presupuesto. La madurez se mide en meses, no en días. Establece el baseline en el primer mes, captura quick wins hacia el tercero e institucionaliza el ciclo de mejora hacia el sexto, cuando el showback evoluciona a chargeback.

Ocho patrones de optimización, del 25 al 45 por ciento sobre el baseline

Los números sostienen el argumento. Para una organización de 500 devs a US$19 por asiento, un baseline de US$9.500 por mes, los pilotos sin disciplina observaron del 50 al 100 por ciento de exceso tras la transición del PRU. Con disciplina aplicada, la misma organización alcanza del 25 al 45 por ciento de reducción sobre el baseline en 60 a 90 días, componiéndose hasta el 40 a 70 por ciento en cargas agénticas. Para organizaciones por encima de 5.000 devs, el ahorro anual supera US$1 millón. El payback es inferior a 60 días.

Ocho patrones entregan esa reducción: enrutamiento dinámico por complejidad, caché jerárquico en tres niveles, subagent fan-out con jerarquía E-mem, plan-then-

execute con un modelo de planificación más económico, caché semántico por similitud de embedding, composición de herramientas en lugar de crecimiento de catálogo, MCP lean con divulgación progresiva de schemas y SDD de agente personalizado guiado por Spec-Kit. Cinco anti-patrones anulan las ganancias y deben irse: el modelo más grande por defecto, un agente donde basta un workflow, el repositorio entero pegado en el contexto, omitir el prompt caching y saltarse el harness.

Azure AI Foundry y migración: PTU versus PAYG

Azure AI Foundry complementa a GitHub Copilot, no lo reemplaza. Copilot resuelve la codificación en la IDE. Foundry corre cargas de producción con gobernanza, un catálogo de más de 1.800 modelos, content safety, RBAC y auditoría dentro del tenant de Azure. Ofrece dos modelos de cobro. PAYG factura por token a las tarifas publicadas sin compromiso, lo que sirve para carga variable o con picos. PTU reserva capacidad de inferencia con throughput garantizado y latencia consistente, lo que sirve para volumen estable y alto. La elección es por workload, no por organización.

La hoja de ruta de adopción de 30, 60, 90 días

La heurística es simple: PAYG por debajo de US$5.000 por mes por modelo, PTU por encima de US$30.000 y un benchmark en el medio. La hoja de ruta sigue el dinero. En los primeros 30 días, analiza el bill preview, configura el cap de paid usage con alertas, enruta modelos en el selector y enciende la telemetría. De los 30 a los 60 días, aplica enrutamiento y caché, cura el copilot-instructions.md y conduce el workshop de tech leads. De los 60 a los 90 días, institucionaliza el showback, conecta el export FOCUS a los dashboards y mueve cargas estables a Foundry donde el compliance o la economía de PTU lo justifiquen.

La conclusión: disciplina, no talento

El setenta y cinco por ciento de las empresas planea IA agéntica. Solo el 34 por ciento reporta transformación profunda. La brecha no es de presupuesto y no es de talento. Es de disciplina aplicada en las seis disciplinas: Cloud, Platform, Context,

Intent, Harness y Token Economy. La economía de tokens es la disciplina que mantiene financiadas a las otras cinco, porque una plataforma que nadie puede costear es una plataforma que nadie opera.

Bajo los PRUs, el costo era por intención, y una sesión agéntica de cuatro horas costaba lo mismo que una pregunta rápida. Bajo tokens, el costo es por trabajo realizado, y cada optimización aparece en la factura. Los equipos que tratan los tokens como recurso finito, con presupuestos por sesión, agente y repositorio, pagan del 30 al 70 por ciento menos por el mismo resultado. Ese margen es la diferencia entre el cementerio y la plataforma, y se compra con práctica, no con headcount.

> DEFINICIÓN
>
> Economía de tokens en una frase: cobra cada llamada de modelo a la tarifa de API publicada en AI Credits, enruta el modelo más pequeño que pasa el eval y financia la plataforma con el ahorro. La disciplina aplicada, no el talento, separa el cementerio de la plataforma.

Referencias

Fuentes primarias para el cambio de facturación, la disciplina de FinOps, las opciones de Azure AI Foundry y la evidencia de adopción.

- GitHub Docs, Usage-based billing and GitHub AI Credits, el cambio de facturación: 1 AI Credit igual a US$0,01, con tokens cobrados a la tarifa de API publicada de cada modelo.
- GitHub Copilot, Available AI models, el selector de modelos al que apunta cada decisión de enrutamiento.
- FinOps Foundation, FinOps Framework, las fases Inform, Optimize y Operate aplicadas al gasto con IA.
- FOCUS, FinOps Open Cost and Usage Specification, el schema estándar para export unificado de costo entre proveedores.

- Microsoft Learn, Azure AI Foundry, la plataforma gobernada para cargas de producción y el catálogo de modelos.
- Microsoft Learn, Provisioned throughput (PTU), la opción de capacidad reservada frente al pay-as-you-go.
- Deloitte, State of AI in the Enterprise, la brecha de adopción: la mayoría de las empresas planea IA agéntica, muchas menos reportan transformación profunda.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps