LEGACY MODERNIZATION FRAMEWORK · PLAYBOOK

# Moderniza un patrimonio COBOL hacia Azure, módulo a módulo, con agentes que puedes gobernar.

La banca, los tributos y los programas sociales de Brasil corren sobre COBOL más antiguo que las personas pagadas para mantenerlo, y la fuerza laboral que lo entiende se va 15% al año. Este playbook es el manual de operación para modernizar un patrimonio mainframe con IA agéntica: una estrategia de portafolio, una arquitectura de referencia gobernada, un guion de siete fases del assess al cutover, y un modelo de fábrica que convierte el primer módulo entregado en producción predecible, todo en GitHub Copilot con el model picker, los AI Credits y el agent mode.


AUTORA

Paula Silva

CARGO

AI-Native Software Engineer

CONTACTO

in/paulanunes


TIEMPO DE LECTURA

~ 45 minutos, de punta a punta

7

FASES DE MODERNIZACIÓN

6

CAPÍTULOS

6

DECISIONES QUE QUEDAN CON HUMANOS

70-
80%

TRABAJO EJECUTADO
POR AGENTES

CAPÍTULOS

Chapter 00

## Quedarse quieto es el mayor riesgo

El patrimonio COBOL de Brasil, una fuerza laboral que se reduce 15% al año, tres fuerzas regulatorias con una ventana de 24 a 36 meses, y cómo la IA agéntica invierte la curva de costo dominada por el entendimiento.

Imperativo →

Chapter 01

## Ninguna estrategia única sirve para todo el parque

El espectro 5R de Gartner leído como portafolio, los cuatro enfoques de Microsoft en Azure, la trampa del JOBOL, y puntuar la preparación para defender Extend sobre Defend.

Estrategia →

Chapter 02

## Cinco capas, modelos enrutados, agentes gobernados

La arquitectura de referencia de cinco capas, el enrutamiento de modelos en el picker de Copilot, el MCP y el copilot-instructions.md como plano de control, y seis agentes personalizados bajo tres patrones de orquestación.

Arquitectura →

Chapter 03

## Evaluar, dividir, traducir, probar el rendimiento

Las cuatro fases operativas que llevan un módulo COBOL de lo desconocido a producción, con los números que separan un módulo entregado de uno estancado.

Guion →

Chapter 04

## Dos mundos corriendo hasta el gate de corte

El strangler fig con CDC y migración gradual de tráfico, shadow traffic y equivalencia conductual, las cinco condiciones de corte, y gobernanza atravesando las tres.

Coexistencia →

Chapter 05

## Del primer módulo a una fábrica

Los seis modos de fallo y la mitigación de cada uno, el modelo de fábrica de reglas de segmentación reutilizables, y un comienzo de dos semanas en GitHub Copilot que un comité directivo puede financiar.

Fábrica →

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 IMPERATIVO

# Quedarse quieto es el mayor riesgo.

La banca central, la recaudación tributaria y los programas sociales de Brasil corren sobre COBOL más antiguo que la mayoría de las personas hoy pagadas para mantenerlo. El código no está fallando. La fuerza laboral que lo entiende se está yendo a una tasa medible, y los reguladores dejaron de esperar. Este capítulo mide el costo de quedarse quieto y muestra por qué la IA agéntica es lo primero que vuelve realista, y no apenas deseable, una ventana de modernización de 24 a 36 meses.

## Brasil corre sobre COBOL, y quienes lo entienden se están yendo

El patrimonio COBOL de Brasil contiene entre 1,25 mil millones y 2,1 mil millones de líneas de código. Está dentro de bancos comerciales, bancos de desarrollo, aseguradoras, organismos gubernamentales y comercio minorista. Estos sistemas no están rotos. Procesan pagos, recaudan impuestos y operan servicios sociales a escala nacional, y lo han hecho de forma confiable durante décadas. Es justamente esa confiabilidad la que vuelve peligroso postergarlos. Un sistema que funciona es fácil de dejar tal cual, y dejarlo tal cual es la decisión que silenciosamente se acumula hasta convertirse en la mayor exposición del balance.

El problema no es el código. Son las personas. Brasil pierde alrededor del 15% de sus desarrolladores COBOL cada año, y no hay un pipeline de reposición detrás de ellos. Las universidades dejaron de enseñar el lenguaje hace una generación. Los ingenieros que escribieron las reglas de negocio originales se están jubilando, y se llevan consigo conocimiento no documentado. Cada año de demora reduce el grupo de personas capaces de explicar qué hace realmente un programa determinado. Por eso trato la deserción de la fuerza laboral, y no la deuda técnica, como el riesgo de primer orden. Se puede refactorizar código malo. No se puede entrevistar a un ingeniero que se fue hace tres años.

## Tres fuerzas regulatorias, y una ventana de 24 a 36 meses

Tres fuerzas regulatorias han convergido, y cada una empuja en la misma dirección. La primera es el mandato de modernización del Banco Central de Brasil para 2028, que exige interfaces de open banking basadas en API y controles de ciberseguridad que las plataformas mainframe no entregan de forma nativa. La segunda es la LGPD, cuyos requisitos de gobernanza de datos (consentimiento, linaje y derecho al olvido) nunca fueron diseñados en arquitecturas construidas en los años 1980. La tercera es la pila de cumplimiento que no deja de subir por debajo de todo: SOX, ISAE 3402 y PCI-DSS. Ninguna de ellas es opcional, y ninguna espera por un ciclo presupuestario conveniente.

Frente a esos plazos está la economía del entendimiento. La ingeniería inversa de las reglas de negocio ha consumido históricamente del 40% al 60% de los presupuestos de modernización. Por eso la mayoría de los proyectos se estanca en la fase de entendimiento y nunca llega a la traducción. Los equipos se quedan sin dinero antes de quedarse sin COBOL. Cuando pongo los plazos regulatorios junto al costo histórico del entendimiento, la ventana útil de acción queda en 24 a 36 meses. No es un horizonte cómodo. Es la razón por la que la IA agéntica importa aquí, porque ataca exactamente la fase que solía devorar el presupuesto.

> DEFINICIÓN
>
> La ventana de acción es de 24 a 36 meses. Está acotada de un lado por el mandato del Banco Central de Brasil para 2028 y la pila de cumplimiento debajo de él, y del otro por una fuerza laboral COBOL que se reduce 15% al año. Quedarse quieto no mantiene la posición. Gasta el horizonte.

## Cómo la IA agéntica invierte la curva de costo dominada por el entendimiento

La razón por la que la modernización de legados tiene un historial tan pobre es adónde iba el dinero. En la distribución tradicional, solo el entendimiento consumía cerca de la mitad del presupuesto: 50% entendimiento, 20% traducción, 15%

pruebas, 15% corte. Los proyectos que gastan la mitad del dinero antes de escribir una línea de código nuevo tienden a no terminar. La IA agéntica invierte esa curva. La distribución orientada por IA queda más cerca de 10% descubrimiento, 20% traducción, 35% pruebas de equivalencia conductual y 35% coexistencia y gobernanza. El presupuesto sale del entendimiento y entra en la validación, que es la fase que de hecho decide si el nuevo sistema es seguro para confiar.

GitHub Copilot en agent mode es lo que vuelve barato el descubrimiento. Impulsado por un modelo de long context del picker, mapea dependencias en miles de archivos, rastrea caminos de ejecución y expone acoplamientos que los desarrolladores experimentados pasan por alto. Los modelos en la clase de 1 millón de tokens sostienen una aplicación COBOL entera en contexto de una sola vez, copybooks, JCL y definiciones de datos juntos. El enrutamiento sigue la forma del trabajo: Haiku 4.5 escanea y hace triaje en alto volumen, Sonnet 5 traduce módulo a módulo, Opus 4.8 hace el análisis arquitectural profundo. Todo eso corre bajo un archivo .github/copilot-instructions.md que fija las convenciones, custom agents que asumen cada etapa del pipeline, y servidores MCP que dan a los agentes acceso vivo al repositorio, al build y al sistema de tickets. El costo se mide en GitHub AI Credits, donde un crédito equivale a un centavo de dólar, así que un techo de créditos por módulo se vuelve un control presupuestario real, y no una conjetura.

## Las seis decisiones que permanecen con humanos

La meta operativa es 70% a 80% del trabajo ejecutado por agentes, con 20% a 30% de supervisión humana. Esa proporción solo se sostiene si la parte humana se gasta en las decisiones con consecuencias irreversibles, y no en revisar sintaxis. Seis decisiones permanecen con las personas, y no se mueven de ahí. Aprobación final de la arquitectura objetivo. Validación de que las reglas de negocio se preservaron, no solo de que el código compiló. Ajuste de rendimiento para patrones de carga específicos. Firma del corte en producción. Autorización de cumplimiento regulatorio. Y escalado de los casos extremos que los agentes señalan pero no pueden resolver por sí solos.

Mantener esas seis con humanos es lo que separa la modernización de la traducción. Gartner es directo sobre el modo de fallo: las herramientas actuales de

GenAI son buenas para traducir código y excluyen explícitamente la arquitectura de su alcance. Traduce COBOL directo a Java sin repensar el diseño y obtienes JOBOL, que es lógica COBOL vestida con sintaxis Java, y perpetúa cada problema estructural del que intentabas escapar. Los agentes hacen el volumen. Los humanos son dueños de la arquitectura, de la postura de cumplimiento y del momento del corte. Esa división del trabajo es el método completo, y el resto de este playbook es cómo ejecutarlo.

> DEFINICIÓN
>
> Seis decisiones nunca dejan a los humanos: aprobación final de la arquitectura objetivo, validación de que las reglas de negocio se preservaron, ajuste de rendimiento para patrones de carga, firma del corte en producción, autorización de cumplimiento regulatorio y escalado de casos extremos. Los agentes ejecutan el otro 70% a 80%. La división es el método.

# Referencias

Fuentes primarias para el riesgo de fuerza laboral, las fuerzas regulatorias y los enfoques de modernización sobre los que se apoya este capítulo.


- Reuters, COBOL Cowboys Aim to Rescue Aging Government Systems, sobre el grupo cada vez menor de ingenieros COBOL y el conocimiento no documentado que se jubila con ellos.
- Banco Central do Brasil (BACEN), origen del mandato de modernización para 2028 y de los requisitos de interfaces de open banking.
- Gartner, 7 Options to Modernize Legacy Systems, el espectro 5R y el argumento por la modernización incremental frente al corte big-bang.
- Microsoft Learn, Mainframe migration in the Cloud Adoption Framework, los enfoques de referencia para rehost, refactor, rearchitect y replace.
- GitHub Copilot coding agent documentation, la plataforma de agent mode que vuelve el descubrimiento automatizado lo bastante barato como para cambiar la curva de costo.

- Grand View Research, Mainframe Modernization Market, contexto de mercado para la escala del patrimonio legado ahora bajo presión de modernización.

---

**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 ESTRATEGIA DE PORTAFOLIO

# Ninguna estrategia única sirve para todo el parque.

Ninguna estrategia de modernización sobrevive al contacto con un parque mainframe entero. Un portafolio COBOL reúne aplicaciones que difieren en criticidad de negocio, acoplamiento, complejidad de datos y vida útil restante. Tratarlas a todas con un solo patrón, sea rehost, refactor o replace, garantiza desperdicio de un lado y riesgo del otro. Este capítulo trata el parque como un portafolio. Mapea cada aplicación a una estrategia, lee los cuatro enfoques de Microsoft como decisiones de herramientas en Azure, nombra la trampa que convierte la traducción en falso progreso, y muestra cómo puntuar la preparación y defender la decisión de Extend sobre el patrón por defecto de Defend.

## Los 5R de Gartner como portafolio, no como mandato

El marco 5R de Gartner define cinco niveles de transformación: Retain, Rehost, Refactor, Rearchitect y Replace. Cada uno carga su propio perfil de costo, riesgo y valor. El error que cometen las empresas es elegir un nivel y aplicarlo al parque entero. El enfoque de portafolio hace lo contrario. Clasifica cada aplicación en dos ejes, criticidad de negocio y complejidad técnica, y la ubica en el patrón que le corresponde. Gartner proyecta que, para 2030, más del 75% de las empresas con mainframes IBM adoptarán estrategias incrementales de bajo riesgo en lugar de reemplazo big-bang.

Recorre el espectro de punta a punta. Retain significa ningún cambio de código: sistemas estables cerca del fin de vida, esfuerzo bajo, valor bajo. Rehost es lift and shift a la nube con el código tal cual, una ganancia rápida de infraestructura. Refactor traduce el lenguaje manteniendo la arquitectura, y es el punto óptimo para la aceleración con IA. Rearchitect rediseña los límites primero y luego traduce, más esfuerzo por más valor. Replace retira el legado por completo y adopta SaaS o

reconstruye. El modelo TIME (Tolerate, Invest, Migrate, Eliminate) aporta la lógica de disposición del portafolio que decide qué nivel merece cada aplicación.

### Dónde entra GitHub Copilot en la clasificación

La clasificación solo es viable si puedes leer el parque rápido. GitHub Copilot en agent mode, impulsando un modelo de long context del picker como Opus 4.8, puede contener una aplicación COBOL entera con sus copybooks, JCL y definiciones de datos de una vez y producir una primera puntuación 5R para cada componente. Codifica la rúbrica de puntuación como un custom agent y un archivo .github/copilot-instructions.md, para que cada repositorio se clasifique igual, no según quien por casualidad lo abra. El arquitecto humano aprueba el bucket. El agente hace la lectura y el primer borrador de la justificación.

## Los cuatro enfoques de Microsoft mapeados en las herramientas de Azure

El marco de Microsoft se alinea con el modelo 5R, pero nombra a Azure como plataforma objetivo. Ofrece cuatro enfoques. Migrate es rehost, entregado con Azure Migrate y emulación de mainframe. Modernize es refactor, entregado con el Mainframe Modernization Agent y Azure AI Foundry. Rearchitect usa Azure Functions, AKS, Service Bus, Event Grid y Cosmos DB. Replace usa Dynamics 365, Power Platform, soluciones ISV y Microsoft Fabric. Junto a estos, el patrón Hybrid y Extend envuelve las transacciones del mainframe en APIs para que el legado siga corriendo mientras se moderniza de forma incremental.

Las herramientas importan menos que el enrutamiento entre ellas. Los conectores MCP dan a un agente de Copilot acceso vivo a las fuentes que necesita para hacer trabajo real: GitHub, Azure DevOps, Backstage, el inventario del mainframe y Key Vault para los secretos. El trabajo concreto bajo cada enfoque es específico. La migración de datos mueve los repositorios VSAM e IMS a Azure SQL. El agendamiento de batch JCL cede lugar a la orquestación moderna. Las transacciones CICS se convierten en REST APIs o Azure Functions. Cada enfoque es un objetivo diferente en la misma plataforma, no un proveedor diferente para generar lock-in.

# La trampa del JOBOL: por qué traducir no es modernizar

La investigación de Gartner (G00822567) confirma un límite que los proveedores tienden a difuminar. Las herramientas actuales de GenAI son excelentes en traducción de código y refactoring de framework, pero excluyen explícitamente la modernización arquitectural de su alcance. Traduce COBOL a Java preservando la estructura original del código y obtienes lo que Gartner llama JOBOL: lógica COBOL implementada en Java. Compila. Pasa una primera mirada. Y perpetúa cada problema estructural del original en un lenguaje nuevo que tu equipo todavía no puede mantener.

Con agentes, el JOBOL ocurre por una razón simple. Un archivo .github/copilot-instructions.md que dice solo convierte este programa COBOL a Java entrega exactamente eso: una transliteración línea a línea. La corrección es ejecutar una pasada de rearquitectura antes. La IA propone los límites objetivo, y los humanos los aprueban. Solo entonces la traducción apunta a esos límites, y no al flujo de control PERFORM THRU original. Gartner también documenta que las herramientas que prometían descomposición automática de monolito en microservicios recularon bastante en esas promesas, así que planifica análisis y diseño manuales sustanciales, sin importar el apoyo de la herramienta. La precisión de traducción en escenarios típicos asistidos por LLM ronda el 80%, y la aceptación de sugerencias de IA se sitúa cerca del 30% en la práctica, así que la revisión humana no es opcional.

> DEFINICIÓN
>
> JOBOL es lógica COBOL vestida con sintaxis Java. Compila, pasa una prueba ingenua y arrastra cada defecto estructural del original a un lenguaje que tus nuevos contratados leen, pero todavía no pueden mantener. Traducir es un paso dentro de la modernización. No es la modernización.

# Puntuar la preparación y defender Extend sobre Defend

Puntúa cada aplicación en cuatro dimensiones antes de asignar una estrategia. La complejidad de código cubre total de líneas, complejidad ciclomática y profundidad de la cadena de llamadas. La complejidad de datos cubre VSAM, IMS, DB2, Adabas e integridad referencial. La complejidad de integración cubre el número de interfaces, el middleware en uso y patrones batch frente a tiempo real. La complejidad operativa cubre agendamiento de batch, disaster recovery, SLAs y exposición a compliance. El Análisis de Puntos de Función, siguiendo el IFPUG, da una medida de tamaño independiente del lenguaje, para que los parques COBOL y Natural se comparen en una única escala. Como referencia, la conversión automática de COBOL a Java corre alrededor de 0,5 a 2 horas por punto de función en código directo, subiendo a 4 a 8 horas en lógica compleja que exige revisión manual.

La puntuación existe para responder a una sola pregunta económica. Gartner (G00823006) separa tres modelos de negocio de GenAI: Defend, que mantiene el legado, Extend, que agrega capacidad a su alrededor, y Upend, que cambia el propio modelo. El dato que debería mover presupuestos es contundente. Las estrategias de Defend muestran un retorno financiero de -52%, las de Extend entregan +134%, y aun así el 50% de la inversión actual en GenAI sigue apuntando a Defend. La lectura es directa. El instinto por defecto es proteger lo que ya funciona, y ese instinto pierde dinero.

### Gobernar el gasto con GitHub AI Credits

El argumento de Extend solo se sostiene si controlas el costo de llegar allí. El uso de agentes de GitHub Copilot se factura en GitHub AI Credits, donde un crédito equivale a un centavo de dólar. Define un techo de créditos por módulo y un gate de validación iterativo antes de cada nueva wave, para que el costo quede acotado por unidad de trabajo en vez de abierto. Una postura de Defend sigue pagando licenciamiento por MIPS y del 60% al 70% del presupuesto de TI en mantenimiento, sin retorno que se acumule. Una postura de Extend gasta créditos medidos en trabajo que produce un activo moderno y mantenible. Orienta el portafolio hacia Extend y Modernize, reserva Defend para las aplicaciones genuinamente cerca del fin de vida, y deja que la puntuación de preparación, no el instinto, decida.

# Referencias

Fuentes primarias para el espectro 5R, el mapeo en Azure, el límite de alcance del JOBOL y la economía de Defend frente a Extend.


- Gartner, 7 Options to Modernize Legacy Systems, el espectro de modernización que este capítulo lee como portafolio y no como mandato.
- Microsoft Learn, Mainframe migration: Cloud Adoption Framework, los cuatro enfoques y sus objetivos de herramientas en Azure.
- Gartner, Predicts 2026: Reducing Costs and Risks in Mainframe and Midrange Modernization, la proyección de que la mayoría de las empresas elige modernización incremental sobre big-bang para 2030.
- IFPUG, Function Point Counting Practices Manual, el método de medición independiente del lenguaje detrás de la puntuación de preparación en COBOL y Natural.
- GitHub Copilot, la plataforma de agentes de referencia para clasificación, traducción y ejecución con costo gobernado en este capítulo.


---


**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 ARQUITECTURA DE REFERENCIA

# Cinco capas, modelos enrutados, agentes gobernados.

Un pipeline de modernización vale tanto como la arquitectura que lo ejecuta. Este capítulo describe la arquitectura de referencia que uso para la modernización agéntica de sistemas legados: cinco capas, de la interfaz del desarrollador al perímetro, una regla de enrutamiento que asigna cada tarea al modelo correcto en el picker de GitHub Copilot, un plano de control formado por conectores MCP y el .github/copilot-instructions.md, y seis agentes personalizados coordinados por tres patrones de orquestación. Cada capa existe para mantener la autonomía dentro de una frontera gobernada.

## Las cinco capas, de la interfaz del desarrollador al perímetro

La arquitectura de referencia tiene cinco capas, y cada una se apoya en la de abajo. La interfaz del desarrollador es donde los humanos conducen el trabajo. Los ingenieros abren GitHub Copilot en agent mode dentro de VS Code o en GitHub.com, y el repositorio lleva un archivo .github/copilot-instructions.md que informa a cada agente las convenciones del proyecto, el lenguaje objetivo y las reglas de modernización. Ese archivo es lo primero que lee un agente, así que ahí es donde empieza el control.

La segunda capa es la orquestación multi-modelo, la inteligencia del pipeline. Un modelo orquestador descompone la tarea, los modelos workers traducen y prueban, los modelos scanners hacen triaje, y Microsoft Semantic Kernel coordina los handoffs. La tercera capa son los conectores MCP, las herramientas empresariales que los agentes alcanzan mediante el Model Context Protocol: GitHub, Azure DevOps, Backstage, la base de datos, la observabilidad, Key Vault y el rastreador de work items. La cuarta capa es la plataforma, donde reside la gobernanza: Azure AI Foundry hospeda los modelos, Microsoft Entra ID gestiona la identidad, Key Vault

guarda los secretos y Azure Monitor registra cada acción. La quinta capa es el perímetro, las interfaces externas: el mainframe del cliente, que todavía conserva la fuente de verdad, los servicios Azure de destino y las APIs externas. El principio de diseño es que toda capa se conecta de vuelta a la capa de plataforma, de modo que ninguna acción de agente escapa de la identidad, el registro o la política.

## Enrutamiento de modelos en el picker de GitHub Copilot

Ningún modelo único maneja bien toda tarea, y ningún modelo único justifica su costo en toda tarea. El picker de GitHub Copilot permite elegir el modelo por tarea, y el pipeline usa una regla de enrutamiento simple: Haiku 4.5 escanea, Sonnet 5 traduce, Opus 4.8 analiza. Haiku 4.5 ejecuta el trabajo de alto volumen y baja complejidad: validación de sintaxis, coincidencia de patrones, clasificación de triaje. Sonnet 5 hace el trabajo de código equilibrado: traducción de COBOL a Java, generación de pruebas, documentación y creación de pull requests. Opus 4.8 asume el razonamiento que necesita el cuadro completo: análisis de aplicación completa, extracción de reglas de negocio y recomendaciones arquitecturales, usando su contexto de 1 millón de tokens para contener una aplicación COBOL entera con copybooks, JCL y definiciones de datos a la vez.

El enrutamiento es una decisión de costo tanto como de calidad. Desde el 1 de junio de 2026, GitHub Copilot cobra el trabajo de los agentes en GitHub AI Credits, donde un crédito vale un centavo de dólar, y los modelos más pesados consumen más créditos por tarea. Enviar una verificación de sintaxis a Opus 4.8 desperdicia créditos; enviar una decisión de arquitectura a Haiku 4.5 desperdicia el análisis. El picker hace la elección explícita y auditable. La misma disciplina se puede automatizar: el Model Router de Azure AI Foundry selecciona un modelo por solicitud a partir de las características de la solicitud, y el material de origen reporta que reduce el gasto de tokens hasta en un 40% al enviar solicitudes simples a modelos más pequeños sin intervención del desarrollador. La regla permanece igual, ya sea que un humano elija el modelo o que lo elija el router: emparejar el modelo con la tarea.

## Conectores MCP y el copilot-instructions.md como plano de control

Los agentes no tocan los sistemas empresariales directamente. Los alcanzan mediante el Model Context Protocol, un estándar abierto que funciona como un conector universal para agentes de IA. Cada servidor MCP expone un conjunto fijo de herramientas, y el agente llama a esas herramientas en lugar de escribir su propio código de integración. El servidor de GitHub lee archivos fuente y abre pull requests. El servidor de base de datos inspecciona esquemas y muestrea datos en modo de solo lectura. El servidor de Key Vault recupera credenciales para que los secretos nunca lleguen al contexto ni a los logs del agente. El servidor de observabilidad compara el rendimiento entre el sistema legado y el moderno. Como MCP es un estándar abierto, servidores de distintos proveedores funcionan con cualquier agente compatible, lo que mantiene la arquitectura libre de lock-in.

La otra mitad del plano de control es el archivo .github/copilot-instructions.md. Es el conjunto persistente de instrucciones que todo agente lee antes de actuar: los estándares de código, la arquitectura objetivo, la regla que prohíbe inyectar cualquier SDK propietario y la restricción de que el código traducido sea código objetivo estándar e idiomático, en lugar de lógica COBOL reescrita en Java. Entre MCP y el archivo de instrucciones, la arquitectura separa lo que un agente puede hacer de lo que tiene permitido hacer. MCP define las acciones alcanzables; el archivo de instrucciones y los niveles de permiso de la plataforma definen las permitidas. Esa separación es lo que hace seguro conceder autonomía al agente.

## Seis agentes personalizados y tres patrones de orquestación

Seis agentes personalizados operan el pipeline, cada uno con un alcance estrecho y un contrato claro de entrada y salida. El agente de Discovery escanea el codebase, empezando por Haiku 4.5 y escalando a Opus 4.8 para el análisis profundo, y produce un grafo de dependencias y un catálogo de reglas de negocio. El agente de Analysis, en Opus 4.8, puntúa cada componente y recomienda una estrategia de modernización. El agente de Translation, en Sonnet 5, convierte un módulo a la vez y abre un pull request con documentación de mapeo. El agente de Test generation, también en Sonnet 5, escribe pruebas de equivalencia conductual y apunta a al

menos 85% de cobertura, con 100% en los caminos críticos de reglas de negocio. El agente de Review coteja el código traducido contra los estándares. El agente de Migration orchestration, en Opus 4.8, coordina a los demás y mantiene el plan.

Tres patrones de orquestación cubren las formas de tarea que surgen, y Microsoft Semantic Kernel implementa los tres de forma nativa. Los handoffs secuenciales mueven el trabajo etapa a etapa, un agente terminando antes de que empiece el siguiente. El patrón de handoff deja que un worker escale de agente a agente cuando encuentra una decisión que no puede resolver. El group chat pone a varios agentes en razonamiento arquitectural colaborativo, consolidado por el orquestador. El peligro contra el que protegen estos patrones es la anarquía de agentes: agentes mal coordinados que colisionan y entran en bucle. La defensa es un orquestador único, un gate de validación entre cada handoff y una acción registrada para cada paso. La autonomía se concede por nivel y se revoca en el primer gate que un agente no logra pasar.

> DEFINICIÓN
>
> La arquitectura en una línea: cinco capas, de la interfaz del desarrollador al perímetro, una regla de enrutamiento (Haiku 4.5 escanea, Sonnet 5 traduce, Opus 4.8 analiza), conectores MCP y .github/copilot-instructions.md como plano de control, y seis agentes personalizados bajo tres patrones de orquestación. Toda capa se conecta de vuelta a la plataforma, así que la autonomía siempre corre dentro de una frontera gobernada.

## Referencias

Fuentes primarias para la arquitectura de referencia, la regla de enrutamiento de modelos, el plano de control MCP y los patrones de orquestación de agentes.


- Microsoft Cloud Adoption Framework, Mainframe migration, la referencia del lado de destino para mapear componentes de mainframe en servicios Azure.
- GitHub, Pick your agent: use Claude and Codex on Agent HQ, el model picker y el agent mode que dirigen la capa de interfaz del desarrollador.

- Model Context Protocol Specification, el estándar abierto detrás de la capa de conectores MCP.
- Microsoft Semantic Kernel, Agent Framework, el framework que implementa los patrones sequential, handoff y group chat.
- Azure AI Foundry, Model Router, selección automática de modelo que la fuente reporta reduce el gasto de tokens hasta en un 40%.
- Azure AI Foundry, Claude models, la capa de plataforma que hospeda los modelos enrutados.
- GitHub Copilot coding agent, el runtime de agente personalizado para tareas autónomas de modernización.

---

**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 DIVIDIR Y TRADUCIR

# Evaluar, dividir, traducir, probar el rendimiento: el guion de ejecución en cuatro fases.

Este capítulo convierte el marco en un guion de ejecución. Cuatro fases llevan un módulo heredado de lo desconocido a producción. Evaluar mapea lo que el código realmente hace. Dividir lo corta en unidades entregables. Traducir lo reescribe módulo por módulo. Rendimiento prueba que el resultado se sostiene bajo carga. Cada fase corre en GitHub Copilot en agent mode, cada una termina en una compuerta de decisión humana, y cada una produce un artefacto listo para rollback. Aquí están las cuatro fases en orden, con los números que separan un módulo entregado de uno estancado.

## Fase 1, Evaluar: el agente de descubrimiento y el grafo de dependencias

El agente de descubrimiento es lo primero que pones en pie. Corre en GitHub Copilot en agent mode, definido como un custom agent a través del .github/copilot-instructions.md, y alcanza el parque mainframe mediante MCP servers: un servidor GitHub para el código fuente, un servidor de base de datos para los schemas, un servidor de observabilidad para los datos de runtime. Ingiere programas COBOL, copybooks, JCL, programas Natural, definiciones CICS y schemas DB2 o Adabas, y luego produce un catálogo de programas, un diccionario de datos, un flujo de jobs JCL, un mapa de transacciones y un puntaje de complejidad de 1 a 10 para cada programa. El escaneo de alto volumen se enruta a Haiku 4.5 en el model picker. El razonamiento profundo de dependencias se enruta a Opus 4.8, cuyo contexto de 1 millón de tokens sostiene miles de programas a la vez, de modo que el agente rastrea acoplamientos que atraviesan todo el parque en lugar de un archivo por vez.

La salida es un grafo de dependencias, y el grafo es donde la evaluación prueba su valor. Separa los huérfanos, programas con casi ninguna dependencia y los

candidatos ideales para empezar, de los programas hub, llamados por muchos otros e incapaces de moverse hasta que sus dependientes estén listos. El análisis estático no es uniformemente confiable, así que el agente etiqueta cada dependencia con un nivel de dificultad: las llamadas CALL directas y las secuencias JCL se resuelven con 95 a 100 por ciento de exactitud, las interacciones CICS LINK y MQ con 80 a 90 por ciento, las llamadas generadas y dinámicas con 50 a 70 por ciento, y el comportamiento que solo aparece en runtime con 35 a 40 por ciento. Para el nivel más difícil, el agente complementa el análisis estático con registros de runtime del SMF, que capturan la CALL dinámica y el enrutamiento indirecto que ninguna pasada estática ve. La compuerta de decisión que cierra esta fase es un portafolio clasificado en los cubos 5R, aprobado por un arquitecto humano.

## Fase 2, Dividir: corte por niveles y el módulo más pequeño entregable

Dividir decide qué va a producción primero. Un parque COBOL bancario típico carga décadas de copybooks compartidos, áreas de datos comunes, llamadas CICS LINK y archivos VSAM tocados por muchos programas. No puedes extraer una pieza sin saber qué toca. El agente de análisis, corriendo en Opus 4.8, lee el grafo de dependencias y encuentra cut-points: fronteras donde los cruces son mínimos, los flujos de datos están bien definidos y son de bajo volumen, y los programas de un lado forman una función de negocio coherente que una API puede encapsular. Los programas dentro de la frontera se vuelven un incremento de modernización. Los programas fuera siguen corriendo en el mainframe.

El orden sigue los niveles de dificultad, no el organigrama. Los módulos hoja de nivel 1, con bajo acoplamiento, se dividen primero, porque construyen la capacidad del equipo sobre victorias que de verdad puedes terminar. Los módulos intermedios de nivel 2 siguen. Los módulos centrales de nivel 3, con alto acoplamiento, quedan al final, cuando el equipo tiene la mayor experiencia y los patrones más reutilizables. El comportamiento de runtime del SMF informa el orden: un módulo que parece aislado en el análisis estático, pero muestra invocación dinámica intensa en runtime, baja en la cola.

### El módulo más pequeño entregable

La unidad que importa es el módulo más pequeño que puede ir a producción por sí solo y probar el pipeline de punta a punta. Para el primer incremento, elige un módulo COBOL de menos de 5.000 líneas, con baja criticidad de negocio, bajo acoplamiento y un cut-point limpio. No es el módulo de mayor valor. Es el que te permite correr descubrimiento, traducción y pruebas de comportamiento hasta una compuerta de decisión firmada, sin una dependencia dura que te bloquee. Ese primer módulo entregado es lo que gana el presupuesto de la siguiente ola, y convierte el plan de secuenciación de una diapositiva en evidencia.

## Fase 3, Traducir: módulo por módulo, la semántica por encima de la sintaxis

La traducción es el paso que todos imaginan cuando oyen modernización, y es el que menos falla. Sonnet 5, el worker de traducción en el model picker, reescribe COBOL en Java, .NET o Python idiomático un módulo a la vez, nunca de golpe. Cada módulo produce un pull request con documentación de mapeo que ata cada unidad traducida de vuelta a su párrafo de origen, y cada pull request va a un revisor humano. Esa es la compuerta de decisión de esta fase: ningún módulo hace merge sin que una persona lea el diff.

La meta es la semántica de negocio, no la estructura sintáctica. El modo de falla tiene nombre, JOBOL: lógica COBOL reimplementada en Java, que compila, pasa una prueba superficial y arrastra a un nuevo lenguaje todos los problemas estructurales del original. Gartner es explícito al afirmar que las herramientas generativas actuales traducen código bien y excluyen la modernización de arquitectura de su alcance, así que la defensa contra el JOBOL es una pasada de rearquitectura antes de la traducción. El agente propone las fronteras objetivo, y un humano las aprueba. Las pruebas de equivalencia de comportamiento se generan a partir del comportamiento del COBOL original, no del código traducido, de modo que la suite valida lo que el sistema debe hacer, y no lo que la traducción produjo por casualidad.

# Fase 4, Rendimiento: baselines de SMF y ninguna regresión más allá de los SLOs

Un módulo modernizado que es correcto pero más lento que el mainframe no va a producción. Los stakeholders de negocio lo van a bloquear, y con razón. Por eso esta fase empieza antes de cualquier traducción, con un baseline de rendimiento construido a partir del propio sistema heredado. Los registros SMF, en particular los tipos 30, 70 a 79 y 110, entregan tiempo de CPU por transacción, espera de I/O y profundidad de cola. Esos datos establecen no solo un promedio, sino la distribución completa: p50, p95 y p99. Un mainframe z16 procesa más de 25.000 transacciones CICS por segundo, y el baseline es el número que el objetivo en Azure tiene que igualar.

El baseline se vuelve un presupuesto de rendimiento con un techo rígido por tipo de carga. Las transacciones online en p95 pueden correr como máximo 10 por ciento más lentas que el baseline, en p99 como máximo 20 por ciento, el tiempo total de batch como máximo 15 por ciento, y el throughput de batch y la latencia de consulta de datos como máximo 10 por ciento. Las pruebas de carga corren con volumen cercano al de producción antes de enrutar cualquier tráfico. Application Insights con tracing distribuido compara el sistema moderno con el baseline de SMF en tiempo real, y si el rendimiento se degrada más allá de la barrera roja, el tráfico vuelve automáticamente al sistema heredado. No se acepta ninguna regresión más allá de los SLOs definidos. La compuerta de decisión es la aprobación de rendimiento por parte del equipo de SRE.

DEFINICIÓN

La vara de go/no-go de cualquier módulo, en una línea: puntaje de complejidad igual o por debajo de 7, cobertura de pruebas igual o por encima de 85 por ciento con 100 por ciento en las rutas de reglas de negocio críticas, 100 por ciento de equivalencia de comportamiento contra el COBOL original, y rendimiento dentro de 10 por ciento del baseline de SMF. El costo permanece acotado por un techo por módulo en GitHub AI Credits, donde 1 credit equivale a un centavo de dólar. Falla en cualquiera de estos y el módulo no va a producción.

# Referencias

Fuentes primarias para las cuatro fases: las herramientas de descubrimiento y traducción, los datos de runtime del mainframe y los patrones de rendimiento de Azure.


- GitHub Copilot coding agent, la referencia para correr los agentes de descubrimiento y traducción en agent mode.
- Model Context Protocol Specification, la interfaz estándar que los agentes usan para alcanzar código fuente, schemas y datos de runtime.
- IBM, System Management Facilities (SMF), los registros de runtime usados para dependencias de nivel difícil y para el baseline de rendimiento.
- Azure Architecture, Strangler Fig Pattern, el patrón de frontera detrás de los cut-points y la fachada de coexistencia.
- Azure Architecture, Reengineer Mainframe Batch Apps, los patrones de rendimiento de batch que el sistema modernizado debe igualar.
- Azure Monitor, Application Insights, validación continua de rendimiento contra el baseline de SMF.
- Gartner, 7 Options to Modernize Legacy Systems, el espectro 5R y la advertencia de que traducir código no es modernizar arquitectura.


---


**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 COEXISTIR Y GOBERNAR

# Dos mundos corriendo hasta el gate de corte.

La parte difícil de la modernización no es traducir COBOL. Es correr el sistema viejo y el nuevo al mismo tiempo, durante meses, sin que los usuarios lo noten, y después probar que el sistema nuevo es lo bastante seguro para asumir el corte. Este capítulo cubre las tres últimas fases del recorrido. Coexist mantiene dos mundos en sincronía. Test los compara bajo carga real. Cutover mueve el tráfico de producción solo cuando se cumplen cinco condiciones. La gobernanza atraviesa las tres, porque los agentes que actúan sin gates, techos y registros son un pasivo, no un acelerador.

## Fase 5, Coexist: el strangler fig, el CDC y la migración gradual de tráfico

La Fase 5 corre el patrón strangler fig. Un enrutador se coloca frente a los dos sistemas y envía cada solicitud al mainframe o a los servicios modernos en Azure. El mainframe sigue siendo la fuente de verdad, con autoridad de lectura y escritura, hasta que un ejecutivo responsable firme el gate de corte. Los servicios modernos corren de forma idempotente junto a él. Tres piezas de infraestructura lo hacen posible. El Azure API Management es la fachada por la que pasa toda solicitud. El Azure Front Door con enrutamiento por peso controla qué porción del tráfico recibe cada sistema, y los pesos cambian sin deploy. El Change Data Capture mantiene consistentes los dos repositorios de datos.

La sincronización de datos es la parte que quiebra proyectos. La regla es simple y absoluta: nunca hagas dual-write. Si los dos sistemas escriben en sus propios repositorios a partir de la misma solicitud, los datos divergen en horas. En cambio, designa un sistema como fuente de verdad para cada dominio de datos y replica hacia el otro. El Change Data Capture lee el log de transacciones del mainframe y transmite los cambios a la base en Azure en tiempo casi real, con una latencia típica

de 1 a 5 segundos. Confluent y tcVISION resuelven la parte específica de mainframe. Para flujos basados en transacción, un patrón orientado a eventos en Azure Service Bus entrega semántica de exactamente una vez. Para datos de referencia que cambian rara vez, una sincronización agendada por API alcanza.

### La migración de tráfico es una escalera, no un salto

El tráfico avanza en peldaños, y cada peldaño tiene que ganarse el siguiente. El sistema moderno empieza con 0% del tráfico real y corre primero en modo shadow, recibiendo copias espejadas de las solicitudes de producción sin impacto para el usuario. Después el tráfico real empieza en 1%. Se expande a 5% solo después de que 1% estuvo estable por al menos 48 horas, y sigue por 10%, 25%, 50% y 100%. Cada peldaño tiene que cumplir los criterios go/no-go antes de que se abra el siguiente. Cuando el tráfico llega a 100%, el mainframe entra en standby y queda disponible por 30 a 90 días para rollback de emergencia. Cada peldaño es reversible en hasta 15 minutos al cambiar las reglas de enrutamiento en API Management, de modo que ningún corte es una puerta de un solo sentido.

## Fase 6, Test: shadow traffic y equivalencia conductual

La Fase 6 es donde se gana la confianza. El shadow traffic es el método. Cada solicitud de producción al mainframe se copia y se envía al sistema moderno al mismo tiempo. La respuesta del mainframe va al usuario. La respuesta del sistema moderno se captura y se compara, nunca se devuelve. Un motor de comparación registra tres cosas para cada solicitud: si las salidas coinciden, cuánto más lento o más rápido está el sistema moderno, y si alguno de los lados produjo un error que el otro no produjo. El shadow traffic tiene que correr por al menos un ciclo de negocio completo, en general un mes, para que aparezcan los patrones diarios, semanales y de cierre de mes. La carga en shadow equivale a la producción completa, así que dimensiona la infraestructura en Azure para eso desde el inicio.

La equivalencia conductual es la prueba de aceptación. La suite de equivalencia se genera a partir del comportamiento COBOL original, no del código traducido, porque las pruebas escritas contra la traducción solo confirmarían sus propios errores. La misma entrada corre por los dos sistemas y las salidas se comparan campo por

campo en miles de fixtures, incluyendo casos de borde, cierre de mes y año bisiesto. La meta es una coincidencia del 100%, con reglas de tolerancia explícitas acordadas de antemano para formatos de fecha y redondeo decimal. El mutation testing verifica las pruebas mismas: inyecta un defecto y, si la suite no lo atrapa, la cobertura no es real. Un mutation score por debajo de 95% marca el módulo para más generación de pruebas. GitHub Copilot en agent mode conduce este trabajo a través del model picker, con Haiku 4.5 escaneando, Sonnet 5 generando pruebas y traducciones, y Opus 4.8 analizando las divergencias que el motor de comparación expone.

## Fase 7, Cutover: cinco condiciones antes del corte

La Fase 7 hace el corte, pero solo cuando las cinco condiciones se cumplen a la vez. Primera, la divergencia del shadow traffic queda por debajo del umbral de SLO a lo largo de un ciclo de negocio completo. Segunda, la paridad de rendimiento se confirma contra la línea base, con el p95 online dentro de 10% y el tiempo total de batch dentro de 15%. Tercera, el procedimiento de rollback fue probado, no solo escrito. Cuarta, el runbook de incidentes está en lugar y el equipo de guardia lo ensayó. Quinta, el ejecutivo responsable firmó la aprobación de cumplimiento. Cuatro de estas condiciones son técnicas y medibles. La quinta es una persona responsable del resultado, y ese es el punto: ningún agente firma un corte.

La decisión go/no-go lee de un checklist fijo, no de una corazonada. Score de complejidad igual o por debajo de 7, cobertura de pruebas igual o por encima de 85%, equivalencia conductual en 100%, p95 online dentro de 10% de la línea base, batch dentro de 15%, cero hallazgos de seguridad críticos o altos, y un ciclo de negocio limpio de shadow traffic. Si cualquier línea falla, el incremento se queda en pruebas. El primer módulo entra en producción de este modo; los siguientes cien se vuelven trabajo de fábrica, con reglas de segmentación reutilizables y pipelines preconstruidos. En régimen estable, un equipo puede mover de 50K a 500K líneas de código por trimestre, con primer resultado en producción en 90 a 180 días, frente a la mediana del sector de 9 a 18 meses.

> DEFINICIÓN
>
> Las cinco condiciones de corte: divergencia del shadow traffic por debajo del umbral de SLO, paridad de rendimiento confirmada contra la línea base, procedimiento de rollback probado, runbook de incidentes ensayado, y firma de cumplimiento de un ejecutivo responsable. Cuatro se miden. La quinta es un nombre en la aprobación.

## Gobernanza: gates de aprobación, techos de costo en AI Credits, registro de auditoría

La gobernanza no es una fase. Atraviesa todas ellas. La meta del modelo operativo es 70% a 80% del trabajo ejecutado por agentes y 20% a 30% de supervisión humana, y seis decisiones permanecen con humanos por más capaces que se vuelvan los agentes: aprobación final de la arquitectura objetivo, validación de que las reglas de negocio se preservaron, ajuste de rendimiento para cargas específicas, firma del corte en producción, autorización regulatoria y escalado de casos de borde. Cada fase termina en un gate de aprobación con un artefacto listo para rollback, y un guardrail verde, amarillo, rojo decide si un incremento puede avanzar.

Tres controles mantienen el trabajo de los agentes rindiendo cuentas. Los techos de costo van primero. El costo de agente en pago por uso escala rápido, así que define un techo por módulo en GitHub AI Credits, donde un crédito equivale a un centavo de dólar, mide cada ejecución de agente contra ese techo y detente al alcanzarlo, en lugar de enterarte por la factura. El registro de auditoría va segundo: cada acción de agente, versión de modelo y justificación de decisión queda registrada, de modo que un corte pueda explicarse y, si hace falta, revertirse. La gobernanza como código va tercero. Un archivo .github/copilot-instructions.md da a cada sesión de Copilot los mismos estándares, los custom agents llevan permisos con alcance, y los servidores MCP exponen los sistemas corporativos por una sola interfaz auditada. Los gates de escalado encaminan los casos difíciles hacia personas: complejidad por encima de 8, cualquier violación de presupuesto de rendimiento, y hallazgos de seguridad por encima de la severidad media. Timeouts, circuit breakers y versionado de agentes contienen los loops descontrolados que Gartner llama anarquía de agentes. Esta es la dirección hacia donde la gobernanza de IA ya avanza, de los principios manuales

hacia controles automatizados alineados con el EU AI Act, el NIST AI Risk Management Framework y la ISO 42001.

# Referencias

Fuentes primarias para la arquitectura de coexistencia, el método de prueba en shadow y los controles de gobernanza de este capítulo.


- Azure Architecture, Strangler Fig pattern, el patrón de referencia para correr sistemas legado y moderno lado a lado durante la coexistencia.
- Confluent, Liberate Mainframe Data, Change Data Capture de bases de mainframe hacia la nube, con una sola fuente de verdad y sin dual-write.
- Azure Front Door, Routing Methods, enrutamiento por peso que mueve el tráfico de 1% a 100% sin deploy.
- Azure API Management, Multi-Region Deployment, la fachada que enruta y espeja solicitudes para el shadow traffic.
- Azure Monitor, Application Insights, tracing distribuido para comparación de rendimiento lado a lado en el corte.
- GitHub Copilot coding agent, el agent mode y el model picker que conducen descubrimiento, traducción y prueba.
- GitHub, Automate repository tasks with agentic workflows, cómo se aplican los gates de aprobación y las verificaciones de calidad en CI/CD.


---


**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 PARTE V, LA FÁBRICA Y EL COMIENZO

# Del primer módulo a una fábrica: qué se rompe, qué escala y por dónde empezar.

Modernizar un patrimonio mainframe no falla en el primer módulo. Falla en el centésimo, cuando el mismo trabajo manual que entregó el módulo uno no puede entregar el módulo cien. Este capítulo nombra los seis modos de fallo que estancan la modernización agéntica, la mitigación de cada uno, el modelo de fábrica que hace predecible la producción y un comienzo de dos semanas en GitHub Copilot que un comité directivo puede financiar.

## Seis modos de fallo y la mitigación de cada uno

He visto la modernización agéntica estancarse siempre de las mismas seis maneras. Ninguna es un fallo de modelo. Son fallos de proceso, y cada uno tiene una mitigación específica que no cuesta nada adoptar el primer día y cuesta mucho instalar después. Nómbralas todas antes de que el primer módulo entre en producción, no después.

El primero es el JOBOL: lógica COBOL traducida línea por línea a Java, de modo que el código nuevo arrastra la estructura vieja y ninguno de los beneficios. La mitigación es una pasada de rediseño arquitectural antes de la traducción, en la que el agente propone los límites objetivo y un arquitecto humano los aprueba. El segundo es el corte big-bang, en el que todo el patrimonio cambia en un fin de semana y no hay camino de vuelta. La mitigación es el patrón strangler fig con change data capture y un período de coexistencia de 12 a 24 meses, para que el tráfico migre de forma gradual y el mainframe siga siendo la fuente de verdad hasta que un ejecutivo responsable firme el corte. El tercero es el test rot, en el que la suite se genera a partir del código traducido y por lo tanto certifica la traducción contra sí misma. La mitigación es generar pruebas de equivalencia conductual a partir del

comportamiento COBOL original y luego hacer mutation testing en la suite para probar que atrapa defectos inyectados.

El cuarto es la anarquía de agentes, en la que varios agentes escriben en el mismo repositorio sin coordinación y sin rastro de auditoría. La mitigación es un orquestador único corriendo en Opus 4.8, toda acción de agente registrada y gates de aprobación en cada punto de decisión. El quinto es el desbordamiento de costos. En GitHub Copilot, el uso de modelos en agent mode se factura en GitHub AI Credits, donde un crédito equivale a un centavo de dólar, y un loop de agente sin límite puede quemar créditos sin nada que mostrar. La mitigación es un techo de créditos por módulo y gates de costo entre las waves, para que la Wave 2 solo empiece cuando la Wave 1 haya probado su economía unitaria. El sexto es el vendor lock-in, en el que el código objetivo queda atado a un SDK propietario. La mitigación es un lenguaje objetivo estándar e idiomático, sin inyección de SDK propietario, y enrutamiento multi-modelo por el picker de Copilot, para que ningún modelo por sí solo sea imprescindible.

## El modelo de fábrica: reglas de segmentación reutilizables y costo predecible

El primer módulo es un trabajo artesanal. Todo módulo posterior debe ser trabajo de fábrica. Ese giro es el objetivo central del marco, y es donde el retorno del esfuerzo realmente aparece. Una vez que un módulo entra en producción con pruebas conductuales que pasan, las reglas de segmentación que lo dividieron, la plantilla de PR que lo documentó y el pipeline de CI/CD que lo validó se convierten en activos reutilizables. Los siguientes cien módulos los heredan.

Codifica esos activos donde el agente los lee. Las reglas de segmentación y las convenciones de traducción pertenecen al .github/copilot-instructions.md, para que cada sesión en agent mode empiece por las mismas reglas en vez de redescubrirlas. El contexto empresarial, el grafo de dependencias, los datos de runtime SMF y las restricciones de compliance llegan al agente por conectores MCP que cada wave reutiliza. Los custom agents encapsulan los roles recurrentes, descubrimiento, traducción, generación de pruebas, revisión, para que el equipo los invoque por nombre y no por prompt. Con eso en su lugar, el costo deja de ser una sorpresa. Cada módulo lleva un presupuesto conocido de GitHub AI Credits, y una fábrica en

régimen estable produce de 50K a 500K líneas de código traducido por trimestre. El primer resultado en producción llega en 90 a 180 días, frente a la mediana del sector de 9 a 18 meses para programas de modernización de legado que nunca se industrializan.

## Un punto de partida de dos semanas en GitHub Copilot

No necesitas una oficina de programa para empezar. Necesitas un módulo y dos semanas. Elige un único módulo COBOL de menos de 5.000 líneas y con baja criticidad de negocio, el tipo de módulo hoja cuyo fallo no detendría al banco. Aprovisiona GitHub Copilot con agent mode y Claude en Azure AI Foundry, para que los mismos modelos del picker de Copilot, Opus 4.8 para análisis, Sonnet 5 para traducción, Haiku 4.5 para escaneo, estén disponibles para el pipeline. Ejecuta el agente de descubrimiento en ese módulo y compara la salida con el modelo mental que tu equipo tiene del código. Los huecos que encuentra, y los que deja pasar, te dicen cuánto confiar en él.

Luego genera la traducción y las pruebas de equivalencia conductual para ese único módulo hoja, y escribe la línea base de gobernanza mientras el trabajo es pequeño: gates de aprobación en cada punto de decisión, un techo de GitHub AI Credits por módulo y registro de auditoría en toda acción de agente, todo registrado en el .github/copilot-instructions.md para que rija en cada sesión posterior. Al cabo de dos semanas tienes un módulo traducido con pruebas conductuales que pasan, un plan de gobernanza documentado y un argumento creíble para la Wave 2. Ese es el artefacto que gana el siguiente compromiso presupuestario.

## Puntos clave para el comité directivo

Un comité directivo no financia una tecnología. Financia un plan ajustado al riesgo. El plan aquí es incremental por diseño: siete fases, un gate de decisión entre cada una y un artefacto con rollback disponible en cada gate, para que el siguiente compromiso presupuestario siempre esté justificado por el resultado anterior. El objetivo operativo es de 70 a 80 por ciento del trabajo ejecutado por agentes y 20 a 30 por ciento de supervisión humana, con seis decisiones que permanecen con

humanos: aprobación final de la arquitectura objetivo, validación de que las reglas de negocio se preservan, ajuste de rendimiento, firma del corte en producción, autorización regulatoria y escalado de casos extremos.

Financia tres salvaguardas y el resto se sigue. Primera, control de costos: un techo de créditos por módulo y gates de costo entre las waves, porque el gasto de agentes en GitHub AI Credits escala rápido y necesita un techo antes que un presupuesto. Segunda, prueba conductual: pruebas generadas a partir del comportamiento COBOL original, no del código traducido, porque una suite que califica la traducción contra sí misma no prueba nada. Tercera, gobernanza en código: gates de aprobación, registro de auditoría y control por orquestador único registrados en el .github/copilot-instructions.md, para que las reglas viajen con el repositorio. Aprueba esas tres y el primer módulo se vuelve una fábrica. Sáltatelas y sigue siendo un experimento.

> DEFINICIÓN
>
> La decisión del comité directivo en una frase: financia un plan incremental de siete fases con un artefacto con rollback disponible en cada gate, mantén las seis decisiones no negociables con humanos y limita el gasto de agentes con un techo de GitHub AI Credits por módulo, para que el primer módulo entregado se vuelva una fábrica repetible y no un caso aislado.

# Referencias

Fuentes primarias para el patrón strangler fig, las herramientas de agentes y los controles de costo en que se apoya este capítulo.


- Gartner, 7 Options to Modernize Legacy Systems, el espectro 5R y la advertencia de que las herramientas actuales de GenAI traducen código pero excluyen la arquitectura, la raíz del modo de fallo JOBOL.
- Microsoft Azure Architecture Center, Strangler Fig pattern, el patrón de coexistencia que mitiga el corte big-bang.
- Microsoft Cloud Adoption Framework, Mainframe migration, la guía de referencia de Microsoft para la migración de mainframe por etapas.

- GitHub Copilot Documentation, About coding agent, el agent mode, la superficie de ejecución del comienzo de dos semanas.
- Microsoft Learn, Use Claude models in Azure AI Foundry, ejecutar los modelos del picker de Copilot vía Foundry para el pipeline de modernización.
- Microsoft Learn, Cost Management and Billing, la superficie de control de costos detrás de los techos por módulo y los gates de costo.
- DORA, 2025 State of AI-assisted Software Development, base empírica sobre la entrega asistida por IA y las salvaguardas que la mantienen estables.

---

**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