

ARQUITECTURA EN LA ERA DE LA IA

El software dejó de obedecer.

La guía de un CTO para diseñar sistemas inteligentes, gobernables y rentables, con el stack Microsoft como referencia.

Roger Carias

FUNDADOR Y CTO DE BITSIDEAS · MICROSOFT CERTIFIED TRAINER

EDICIÓN 2026

Por qué escribí este libro

Llevo más de dieciocho años diseñando sistemas sobre el stack de Microsoft. Empecé cuando SharePoint se instalaba en servidores propios y una "integración" era un servicio de Windows que copiaba archivos a medianoche. Desde entonces he visto pasar varias olas que prometían cambiarlo todo: SOA, la nube, los microservicios, el low-code. Cada una cambió algo. Ninguna cambió tanto como la que estamos viviendo.

La diferencia no está en la potencia de los modelos, aunque es enorme. Está en que, por primera vez, ponemos en el centro de nuestros sistemas un componente que no se comporta igual dos veces. Toda la disciplina de la arquitectura de software se construyó sobre una promesa implícita: la misma entrada produce la misma salida. Esa promesa se rompió, y con ella una buena parte de nuestras herramientas mentales.

En los últimos años, desde Bitsideas, me ha tocado acompañar proyectos de IA en salud, construcción, seguridad y salud en el trabajo, educación y servicios. Casi todos empezaron igual: una demo que impresionó a la junta directiva en una tarde. Y casi todos se encontraron con el mismo muro unas semanas después. El problema rara vez fue el modelo. Fueron los datos que nadie había ordenado, los permisos que nadie había pensado, el costo que nadie había calculado y la pregunta que nadie supo responder cuando algo salió mal: *¿por qué hizo eso?*

Este libro es lo que me habría gustado tener en la mano al comienzo de esos proyectos. No es un manual de producto ni un catálogo de servicios de Azure. Los productos cambian de nombre cada seis meses (lo verá varias veces en estas páginas). Lo que dura son los criterios: cómo separar lo determinista de lo probabilístico, cuánta autonomía darle a un agente y con qué evidencia, dónde poner la frontera de confianza, cómo medir lo que antes simplemente se probaba, cuánto cuesta realmente una conversación.

Escribo desde una posición concreta, y conviene declararla. Soy Microsoft Certified Trainer y mi consultora es partner de Microsoft. Por eso el libro aterriza los principios en ese stack: Foundry, Copilot Studio, Agent Framework, Fabric, Entra, Purview. Pero he intentado ser honesto con sus límites, mencionar las alternativas de AWS, Google y el mundo open source cuando son mejores para un caso, y mirar con atención lo que hace Apple con la inferencia en el dispositivo, porque su forma de pensar la privacidad tiene lecciones que cualquier arquitecto debería estudiar.

También escribo desde Latinoamérica, y eso pesa más de lo que parece. Aquí las regiones de nube son menos, las regulaciones de IA van llegando a destiempo, los presupuestos son más cortos y los equipos más pequeños. Muchos libros sobre arquitectura de IA asumen una empresa de Silicon Valley con cien ingenieros de plataforma. Este asume una organización que necesita resultados en un trimestre, con un equipo que también tiene que mantener lo que ya existe.

Una advertencia sobre las cifras. Todo dato externo lleva su referencia, y he procurado que sean fuentes primarias. Los precios que aparecen en los ejemplos de costo son ilustrativos y están marcados como tales; la aritmética sí es exacta. Los estados de los productos (disponibilidad general o preview) corresponden a septiembre de 2026 y van a cambiar. Los casos de estudio son compuestos: combinan patrones de proyectos reales sin exponer a ningún cliente.

Si al terminar el libro su equipo discute menos sobre qué modelo usar y más sobre qué evidencia necesita para dejar que un agente actúe solo, habrá cumplido su propósito.

Roger Carias

Bogotá, septiembre de 2026

Cómo leer este libro

El libro tiene cuatro partes y se puede leer en orden o por saltos. La **Parte I** plantea el cambio de fondo y los principios que uso para tomar decisiones; si solo tiene una tarde, lea los capítulos 1 y 3. La **Parte II** recorre los patrones que aparecen en casi cualquier sistema con IA: datos, recuperación, agentes, interoperabilidad y la decisión de dónde ejecutar la inferencia. La **Parte III** trata los atributos de calidad que separan una demo de un sistema en producción: seguridad, gobierno, evaluación y costo. La **Parte IV** lo junta todo en una arquitectura de referencia, cuatro casos y una hoja de ruta para la organización.



Mapa del libro — Las cuatro partes y sus dependencias. El capítulo 13 integra todo lo anterior en un solo plano.

Cada capítulo incluye un **modelo propio**, una herramienta de decisión con nombre que puede llevar a una reunión de arquitectura. Los recuadros siguen una convención fija:

VISIÓN DE CTO

Mi criterio personal sobre un dilema. Es opinión, y está marcada como tal.

EN EL STACK MICROSOFT

Cómo se implementa la idea con productos concretos, con su estado a septiembre de 2026.

ANTIPATRÓN

Un error que he visto repetirse, y lo que cuesta.

Al final de cada capítulo hay un resumen de ideas y una lista de preguntas para trabajar con su equipo. Las referencias aparecen numeradas al cierre de cada capítulo. Los apéndices traen plantillas (ADR, checklist de revisión, canvas de caso de uso, evaluación de riesgo) y un glosario con más de cien términos.

01

PARTE I

El cambio de paradigma

Qué se rompe cuando un componente probabilístico entra al núcleo de un sistema, cómo está el terreno en 2026 y con qué principios decidir.

CAPÍTULOS

01	Cuando el software dejó de ser determinista	8
02	El mapa del territorio en 2026	20
03	Principios de arquitectura para sistemas con IA	31

01

Cuando el software dejó de ser determinista

Durante cincuenta años diseñamos sistemas bajo un supuesto tan obvio que nadie lo escribía: la misma entrada produce la misma salida. Meter un modelo de lenguaje en el núcleo de un sistema rompe ese supuesto, y con él buena parte de las herramientas mentales del arquitecto. Este capítulo trata de qué se rompe exactamente, qué atributos de calidad aparecen y por qué el trabajo del arquitecto pasa de especificar comportamiento a diseñar restricciones.

EN ESTE CAPÍTULO

- 1.1 Mil respuestas a la misma pregunta
- 1.2 Qué cambia cuando el núcleo es probabilístico
- 1.3 Atributos de calidad que no estaban en la lista
- 1.4 Del sistema de registro al sistema de razonamiento
- 1.5 Breve historia: del monolito al sistema nativo de IA
- 1.6 El espectro de autonomía
- 1.7 El arquitecto como diseñador de restricciones

1.1 Mil respuestas a la misma pregunta

En septiembre de 2025, un equipo de Thinking Machines Lab publicó un experimento que debería estar colgado en la pared de cualquier oficina de arquitectura. Le pidieron a un modelo abierto de 235.000 millones de parámetros la misma cosa mil veces, con temperatura cero, que es la configuración que la mayoría de los equipos usa cuando quiere "respuestas deterministas". Obtuvieron 80 completados distintos. Los primeros 102 tokens fueron idénticos en todos los casos; en el token 103 el texto empezó a bifurcarse. La causa principal no era el azar del muestreo sino algo mucho más prosaico: la carga del servidor. Según cuántas peticiones compartan el mismo lote en la GPU, cambian los órdenes de las operaciones de punto flotante, y con ellos el resultado. Los autores lo llamaron falta de *invariancia de lote*.

La consecuencia arquitectónica es enorme: parte de la variabilidad viene de la infraestructura compartida en la que corre el modelo, y esa infraestructura no es nuestra. Microsoft lo deja por escrito en la documentación del parámetro `seed` de Azure OpenAI: aun con la misma semilla y la misma huella de backend (`system_fingerprint`), el determinismo no está garantizado y es habitual observar cierta variación, sobre todo en respuestas largas.

La segunda fuente de variación es más lenta y más peligrosa. En 2023, Lingjiao Chen, Matei Zaharia y James Zou midieron el comportamiento de GPT-4 en dos fechas separadas por tres meses. En una tarea concreta (decidir si un número es primo o compuesto) la exactitud pasó de 84 % en marzo a 51 % en junio, sin que los usuarios cambiaran una sola línea de código. El "mismo" servicio ya no era el mismo. Su conclusión, publicada después en *Harvard Data Science Review*, fue que cualquier sistema que dependa de un LLM necesita monitoreo continuo de su comportamiento, no sólo de su disponibilidad.

Hay una tercera fuente, y es contractual. En Microsoft Foundry, un modelo GA tiene fijada desde su lanzamiento una fecha de retiro a 18 meses; a los 12 meses deja de estar disponible para clientes nuevos, y cuando llega la fecha las peticiones devuelven `410 Gone`. Las implementaciones estándar se actualizan automáticamente a la versión de reemplazo, región por región, salvo que el equipo lo desactive; las de throughput aprovisionado no se actualizan solas y hay que migrarlas a mano. Dicho de otro modo: el componente más importante del sistema tiene fecha de caducidad impresa, y su sucesor no se comportará igual.

Muestreo, infraestructura, versión. Tres fuentes de no determinismo, de las cuales el equipo controla parcialmente una. Ése es el punto de partida.

1.2 Qué cambia cuando el núcleo es probabilístico

El software clásico se piensa como una función. Dada una entrada válida, el sistema produce una salida correcta o falla de forma reconocible: una excepción, un código HTTP 500, un timeout. Las pruebas unitarias funcionan porque podemos escribir `assertEquals(esperado, obtenido)`. Los contratos de API funcionan porque un esquema dice qué entra y qué sale. Incluso en los sistemas distribuidos más caóticos el caos viene de la composición, no de la pieza.

Un modelo de lenguaje no es una función en ese sentido. Es una distribución de probabilidad sobre secuencias de texto, condicionada por un contexto que incluye instrucciones, historial, documentos recuperados y resultados de herramientas. Cuando lo invocamos, extraemos una muestra de esa distribución. La salida puede ser excelente, mediocre o plausible pero falsa, y las tres llegan con el mismo código HTTP 200 y la misma seguridad en el tono. El modelo no lanza excepciones cuando se equivoca. Ése es el cambio más profundo: **el fallo deja de ser una señal y pasa a ser un dato que hay que medir**.

La guía de Azure Well-Architected Framework para cargas de IA lo formula de manera sobria: estas cargas sustituyen funcionalidad determinista por comportamiento no determinista en partes del sistema, precisamente donde los resultados fijos son poco prácticos. La frase importante es "en partes". Nadie sensato construye un sistema entero sobre un modelo. Lo que se construye es un sistema determinista con islas probabilísticas, y el trabajo de arquitectura consiste en decidir dónde van esas islas, qué tan grandes son y qué diques las rodean.

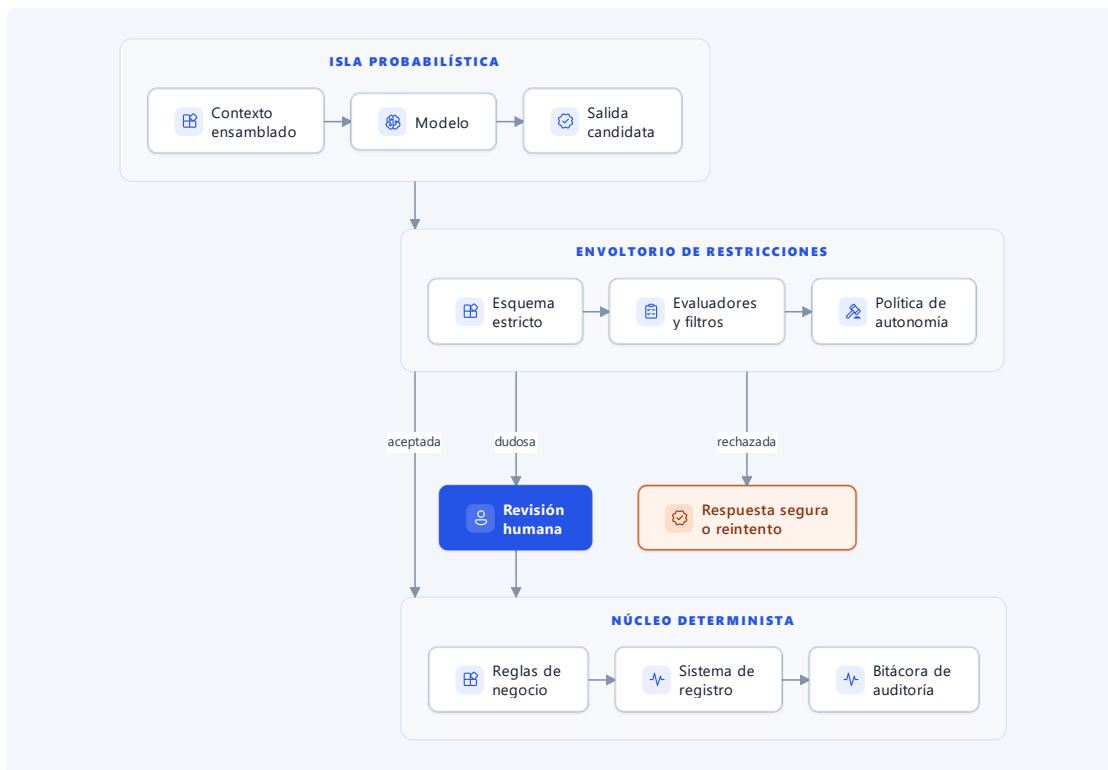


Figura 1.1 — Un sistema con IA bien diseñado es un núcleo determinista con islas probabilísticas acotadas. La salida del modelo es siempre una candidata; el envoltorio decide qué pasa con ella.

La figura 1.1 resume la postura de este libro. La salida del modelo nunca entra directamente al sistema de registro. Pasa por un envoltorio que valida forma (esquema), contenido (evaluadores, filtros de seguridad, chequeos de fidelidad) y permiso (qué tanto puede hacer el sistema sin un humano). Sólo entonces se convierte en una acción de negocio. En una demo parece burocracia. En producción separa un incidente menor de una noticia.

ANTIPATRÓN

Fijar `temperature=0` y declarar el sistema "determinista". Reduce la variación del muestreo, pero no toca la de infraestructura ni la de versión del modelo. Lo he visto en varios proyectos: el equipo aprueba la solución con 40 casos de prueba en la máquina de un desarrollador, la pasa a producción, y tres semanas después nadie entiende por qué el mismo documento se clasifica distinto el lunes y el jueves. Las pruebas de igualdad exacta sobre texto generado son un falso consuelo; se necesitan evaluaciones estadísticas sobre conjuntos de casos, tema que desarrollamos en el capítulo 11.

Del contrato de salida al contrato de comportamiento

En un servicio clásico, el contrato es la firma: tipos de entrada, tipos de salida, códigos de error. Con un modelo, ese contrato sigue siendo necesario pero ya no alcanza. Hoy podemos forzar la forma: las *structured outputs* de Azure OpenAI obligan a que la respuesta cumpla un JSON Schema cuando se activa el modo estricto, a diferencia del viejo modo JSON, que sólo garantizaba JSON válido. Eso resuelve el problema del parser. No resuelve que el campo `monto_aprobado` contenga una cifra inventada con el formato correcto.

Lo que hace falta es un contrato de comportamiento: una especificación estadística de qué debe hacer el componente, medida sobre una población de casos. Algo así como "en el conjunto dorado de 600 consultas, la fidelidad media es $\geq 4,2$ sobre 5, ninguna respuesta cita un documento al que el usuario no tiene acceso, y el 95 % de las respuestas llega en

menos de 6 segundos". Ese contrato se versiona, se revisa en cada cambio de prompt o de modelo y bloquea despliegues cuando se incumple. Es el equivalente funcional de una suite de pruebas, y quien no lo tiene está desplegando a ciegas.

1.3 Atributos de calidad que no estaban en la lista

La norma ISO/IEC 25010, la referencia clásica para hablar de atributos de calidad, se revisó en 2023. Entre otros cambios, la seguridad física y operacional (*safety*) entró como característica de primer nivel, la usabilidad pasó a llamarse capacidad de interacción y la portabilidad se convirtió en flexibilidad. Son ajustes razonables, pero la norma sigue pensada para software cuyo comportamiento se puede especificar de antemano. Cuando el núcleo es probabilístico aparecen atributos que no tienen casilla, y que el arquitecto tiene que poner sobre la mesa desde la primera reunión de requisitos.

Fidelidad (*groundedness*). Mide si la respuesta se sostiene en las fuentes que se le dieron al modelo o si agrega afirmaciones de su propia cosecha. Es el atributo que más le importa a un usuario de negocio, aunque nunca lo nombre así. Microsoft Foundry trae un evaluador de *groundedness* que puntúa de 1 a 5 usando un modelo como juez, y una variante *Groundedness Pro*, todavía en preview, que da un veredicto binario apoyándose en Azure AI Content Safety. La fidelidad no es exactitud: una respuesta puede ser fiel a un documento desactualizado y estar equivocada.

Deriva. El comportamiento del sistema cambia sin que nadie toque el código. Puede venir del modelo (actualización de versión), de los datos (el índice de búsqueda envejece, entra una política nueva que contradice a la anterior) o de los usuarios (empiezan a preguntar cosas para las que el sistema no fue pensado). Las guías de Well-Architected para IA piden detectar y mitigar la degradación de forma automatizada, con alertas cuando las respuestas empiezan a desviarse de lo esperado de manera sostenida. En la práctica significa ejecutar evaluaciones programadas sobre tráfico real muestreado, no sólo antes de cada despliegue.

Costo por unidad de trabajo. En un sistema clásico el costo marginal de una petición es casi cero; se paga capacidad. Con modelos se paga por token, y el número de tokens depende de decisiones de diseño que a veces toma el propio modelo: cuántas veces llama a una herramienta, cuánto "razona" antes de contestar, cuánto contexto se recupera. El costo deja de ser un dato de infraestructura y se vuelve un atributo funcional con varianza. Lo vemos a fondo en el capítulo 12, pero conviene un orden de magnitud ya.

Supongamos un asistente interno con 50.000 conversaciones diarias, seis llamadas al modelo por conversación, 3.000 tokens de entrada y 400 de salida por llamada, a precios de lista ilustrativos de US\$2 por millón de tokens de entrada y US\$8 por millón de salida (verificar en la calculadora de Azure; los precios cambian). Son 300.000 llamadas al día: 900 millones de tokens de entrada (US\$1.800) y 120 millones de salida (US\$960), es decir US\$2.760 diarios, unos US\$82.800 en un mes de 30 días. Si un cambio de prompt hace que el agente pase de seis a nueve llamadas por conversación, el costo sube 50 % sin que haya cambiado el número de usuarios. Ese cambio de prompt probablemente lo aprobó alguien que no mira la factura.

La tendencia de precios ayuda, pero no tanto como parece. El AI Index 2025 de Stanford estimó que el costo de inferencia para un sistema con desempeño equivalente a GPT-3.5 cayó más de 280 veces entre noviembre de 2022 y octubre de 2024. Al mismo tiempo, los modelos de razonamiento y los agentes multiplican los tokens por tarea. El precio unitario baja; el consumo por caso de uso suele subir más rápido.

Autonomía. Es el atributo nuevo más importante y el menos discutido en los documentos de requisitos. Describe cuánto puede hacer el sistema sin que un humano intervenga: sugerir, redactar, ejecutar, encadenar acciones, fijar sus propios subobjetivos. Lo decide el diseño, no el modelo. Le dedicamos la sección 1.6 completa.

Completan la lista la *latencia variable*, que depende de la longitud de la salida y de los pasos del agente, y la *reemplazabilidad del modelo*: cuánto cuesta cambiarlo cuando se retire. La tabla 1.1 los compara con sus parientes clásicos.

Tabla 1.1 — Atributos de calidad clásicos frente a los que introduce un componente probabilístico

Atributo clásico	Atributo en sistemas con IA	Cómo se mide	Qué lo rompe
Corrección funcional	Fidelidad y exactitud	Evaluadores sobre conjunto dorado; muestreo de producción con juez humano o LLM	Recuperación pobre, fuentes contradictorias, prompt ambiguo
Estabilidad de versión	Deriva de comportamiento	Evaluaciones programadas; comparación contra línea base por versión de modelo	Actualización automática del modelo, índice envejecido, cambio en la población de usuarios
Costo de capacidad	Costo por tarea completada	Tokens por conversación y por tarea exitosa; percentiles, no promedios	Bucles de agente, contexto inflado, razonamiento excesivo
Rendimiento	Latencia variable	Tiempo al primer token y tiempo total, p50/p95 por tipo de tarea	Salidas largas, cadenas de herramientas, throttling por cuota
Control de acceso	Autonomía acotada	Acciones ejecutadas sin aprobación, por nivel de riesgo	Permisos amplios de herramientas, ausencia de política de escalamiento
Portabilidad	Reemplazabilidad del modelo	Esfuerzo de migración medido en el último cambio de versión	Prompts acoplados a un modelo, ausencia de evaluaciones reutilizables

VISIÓN DE CTO

Si un documento de arquitectura para un sistema con IA no tiene una sección con objetivos numéricos de fidelidad, costo por tarea y nivel de autonomía, no está terminado. Mi recomendación es no aprobarlo en el comité. Suena duro, pero es mucho más barato discutir esos números en la fase de diseño que reconstruirlos después de un incidente, cuando ya hay usuarios que dejaron de confiar en la herramienta.

1.4 Del sistema de registro al sistema de razonamiento

En 2011, Geoffrey Moore propuso en un informe para AIIM una distinción que se volvió vocabulario común: los *sistemas de registro* (ERP, contabilidad, nómina, la verdad transaccional de la empresa) frente a los *sistemas de interacción* (correo, colaboración, redes, todo lo que gira alrededor de las personas y sus conversaciones). Durante una década la arquitectura empresarial se organizó alrededor de esa frontera: un núcleo transaccional estable y una periferia ágil que lo consumía por API.

Los modelos de lenguaje agregan una tercera categoría, que en este libro llamaremos **sistema de razonamiento**. No guarda la verdad (eso sigue siendo trabajo del sistema de registro) ni se limita a mostrarla. Interpreta una intención expresada en lenguaje natural, decide qué información necesita, la busca, la combina con reglas que a veces están escritas y a veces no, y propone o ejecuta una acción. Hace lo que antes hacía un analista con tres pestañas abiertas y una hoja de cálculo.

Andrej Karpathy le puso nombre a la consecuencia para el oficio en su charla de junio de 2025 en la AI Startup School de Y Combinator: después del *Software 1.0* (código escrito a mano) y del *Software 2.0* (pesos de redes neuronales aprendidos de datos) llega un *Software 3.0* en el que los prompts en lenguaje natural funcionan como programas. Se puede discutir la numeración. Lo que no se puede discutir es que en muchas empresas ya hay lógica de negocio crítica viviendo en un párrafo de instrucciones que nadie versiona, nadie prueba y nadie revisa en el comité de cambios.

El sistema de razonamiento tiene tres rasgos que lo distinguen. Su entrada es ambigua (una pregunta, un correo, un PDF escaneado) en lugar de un formulario validado. Su salida es una propuesta con un grado de confianza implícito, no un resultado. Y su comportamiento depende del contexto que se le ensambla en cada invocación, de modo que la calidad del sistema es, en buena medida, la calidad de su capa de datos y recuperación (el capítulo 4 y el capítulo 5 tratan esto en detalle). De ahí sale un principio que repetiremos: el sistema de razonamiento propone, el sistema de registro dispone. Las escrituras sobre la verdad transaccional pasan por reglas deterministas, con identidad y auditoría, aunque la decisión la haya sugerido un modelo.

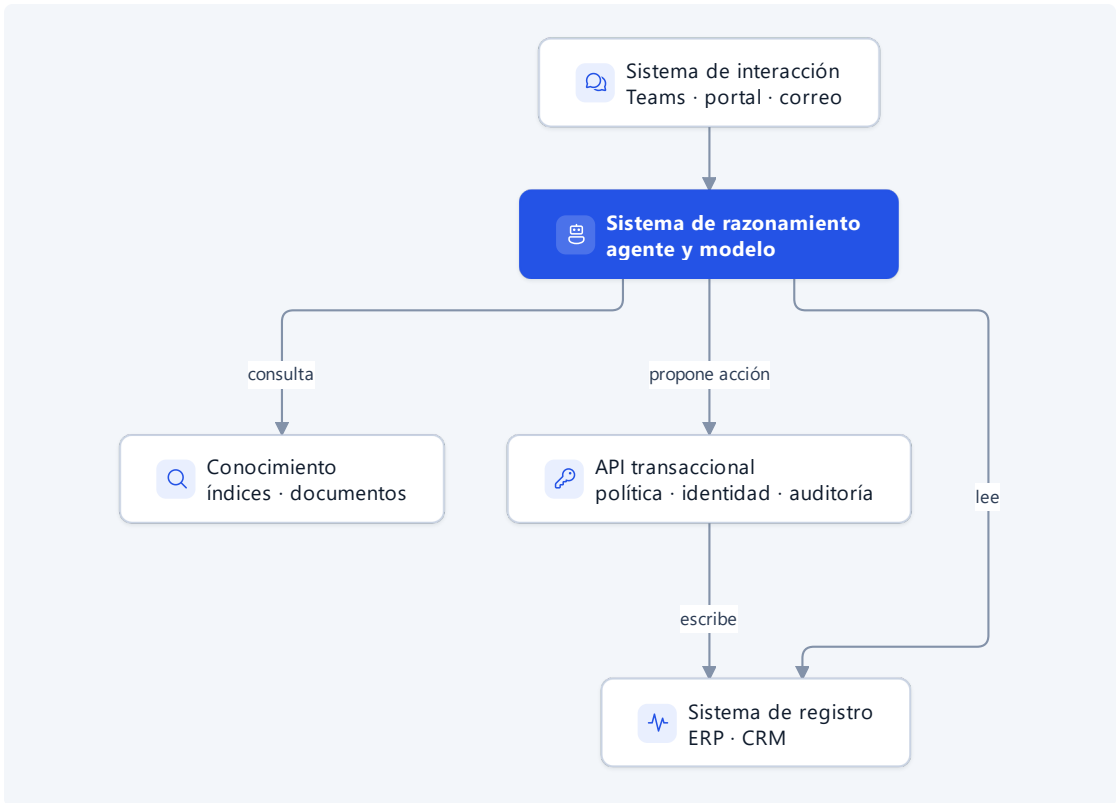


Figura 1.2 — El sistema de razonamiento se sitúa entre la interacción y el registro. Lee con libertad, pero escribe sólo a través de APIs con política, identidad y auditoría.

La figura 1.2 muestra una topología razonable como punto de partida. El agente tiene acceso de lectura amplio (dentro de los permisos del usuario, siempre) y acceso de escritura estrecho, mediado por APIs que ya existían o que se crean para el caso. Esas APIs no confían en el agente más de lo que confiarían en un formulario web: validan, aplican reglas y registran quién pidió qué. Si la empresa no tiene esas APIs, el primer proyecto de IA casi siempre termina siendo, en la práctica, un proyecto de integración. No es glamoroso. Es la mitad del trabajo.

1.5 Breve historia: del monolito al sistema nativo de IA

Cada salto arquitectónico de los últimos treinta años respondió a una fuerza dominante distinta, y conviene recordarlo para no confundir lo que cambia con lo que se mantiene.

El **monolito** optimizaba la simplicidad de despliegue. La **arquitectura orientada a servicios** (SOA) respondió a la integración empresarial con contratos formales, buses y mucha ceremonia. Los **microservicios** respondieron a la velocidad de cambio, a cambio de complejidad operativa. Lo **nativo de la nube** respondió a la elasticidad y convirtió la operación en software.

En todos esos saltos el componente individual siguió siendo determinista. Cambió la granularidad, el acoplamiento y la forma de desplegar, pero un servicio de facturación de 2018 hacía exactamente lo que su código decía. Lo **nativo de IA** es el primer salto en el que la naturaleza del componente cambia, no sólo su tamaño o su ubicación. Por eso la analogía de "es otro microservicio más" falla tan rápido.

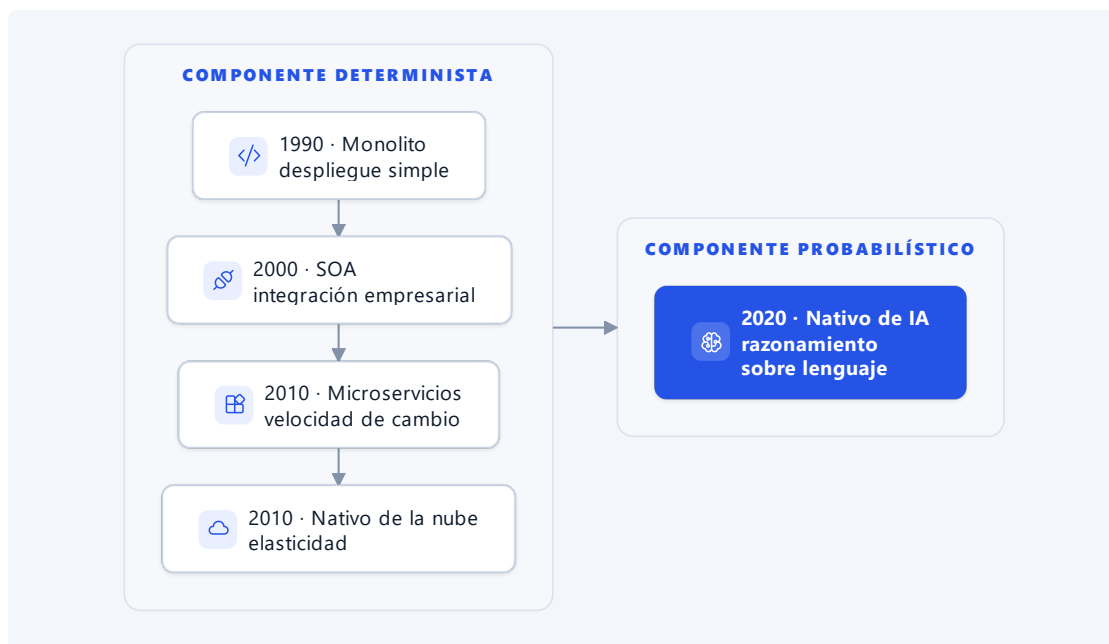


Figura 1.3 — Cada etapa respondió a una fuerza distinta, pero en las cuatro primeras el componente siguió siendo determinista; lo nativo de IA cambia su naturaleza. Las fechas son aproximadas; en la mayoría de las empresas las cinco conviven hoy.

Dos matices. Primero, ninguna etapa reemplazó a la anterior; las acumuló. Cualquier empresa mediana de la región tiene hoy un ERP monolítico, un bus de integración de la época SOA, algunos microservicios y funciones serverless, y ahora agentes. El arquitecto no diseña en una hoja en blanco. Segundo, lo nativo de IA hereda todo lo aprendido: los agentes necesitan contratos (esta vez con forma de esquemas de herramientas y protocolos como MCP, que vemos en el capítulo 7), observabilidad distribuida, despliegues seguros y gobierno de identidades. Quien no dominó lo nativo de la nube va a sufrir el doble con lo nativo de IA, porque los problemas viejos siguen ahí y se suman los nuevos.

NOTA

"Nativo de IA" no significa "todo con IA". Un sistema nativo de IA es aquel cuya arquitectura asume desde el diseño que habrá componentes probabilísticos, y por eso incorpora evaluación, trazabilidad, envoltorios de restricción y política de autonomía como piezas de primera clase. Puede tener un solo modelo y cien servicios deterministas.

1.6 El espectro de autonomía

La pregunta más útil que se le puede hacer a cualquier propuesta de IA no es "¿qué modelo usa?" sino "¿qué puede hacer sin preguntarle a nadie?". La respuesta determina el riesgo, los controles necesarios, el costo de un error y, muchas veces, el valor del caso de uso.

La industria automotriz resolvió un problema parecido hace años con la norma SAE J3016, que define seis niveles de automatización de la conducción, del 0 al 5, según qué parte de la tarea asume el sistema y qué parte el conductor. En el mundo de los agentes, Kevin Feng, David McDonald y Amy Zhang propusieron en 2025 cinco niveles definidos por el rol del usuario (operador, colaborador, consultor, aprobador, observador) y defendieron una idea de fondo: el nivel de autonomía de un agente es una decisión de diseño deliberada, separada de su capacidad. Un modelo muy capaz puede y a menudo debe operar con poca autonomía. Karpathy usa la imagen de un "deslizador de autonomía" que el producto expone y el usuario ajusta.

El modelo que sigue aterriza esas ideas en un espectro pensado para quien tiene que aprobar una arquitectura empresarial. La diferencia con los marcos citados es que cada nivel lleva asociados los controles mínimos que lo hacen defendible ante un comité de riesgos.

MODELO**Espectro de Autonomía Acotada (EAA)**

Seis niveles, de A0 a A5. Cada nivel describe qué hace el sistema sin intervención humana y qué controles mínimos exige. Un mismo producto puede tener funciones en niveles distintos; el nivel se asigna por acción, no por aplicación.

- 1. A0 · Determinista** — Sin modelo en la ruta de decisión. Reglas, flujos y código. Sigue siendo la opción correcta más a menudo de lo que se admite.
- 2. A1 · Asistido** — El modelo sugiere (resume, redacta, clasifica) y la persona decide y ejecuta. Controles: citas a la fuente, etiqueta visible de contenido generado, evaluación de fidelidad previa al despliegue.
- 3. A2 · Copiloto** — El modelo prepara la acción completa (borrador de correo, asiento contable, ticket) y la persona la aprueba con un clic. Controles: vista previa obligatoria, diferencia visible entre lo propuesto y lo existente, registro de quién aprobó.
- 4. A3 · Delegado** — El sistema ejecuta acciones de bajo riesgo y reversibles sin aprobación, dentro de límites explícitos (monto, alcance, tipo de dato), y escala el resto. Controles: lista cerrada de herramientas, identidad propia del agente con menor privilegio, umbrales de escalamiento, reversibilidad probada.
- 5. A4 · Autónomo supervisado** — El agente persigue un objetivo, planifica y encadena pasos; los humanos supervisan por muestreo y excepción. Controles: presupuesto de tokens y de pasos por tarea, trazas completas, evaluación continua sobre producción, botón de parada, revisión periódica de acciones.
- 6. A5 · Autónomo pleno** — Sin supervisión humana regular, el sistema fija subobjetivos y actúa. En entornos empresariales regulados, a septiembre de 2026, es difícil justificarlo fuera de dominios muy acotados. Está aquí para marcar el límite.

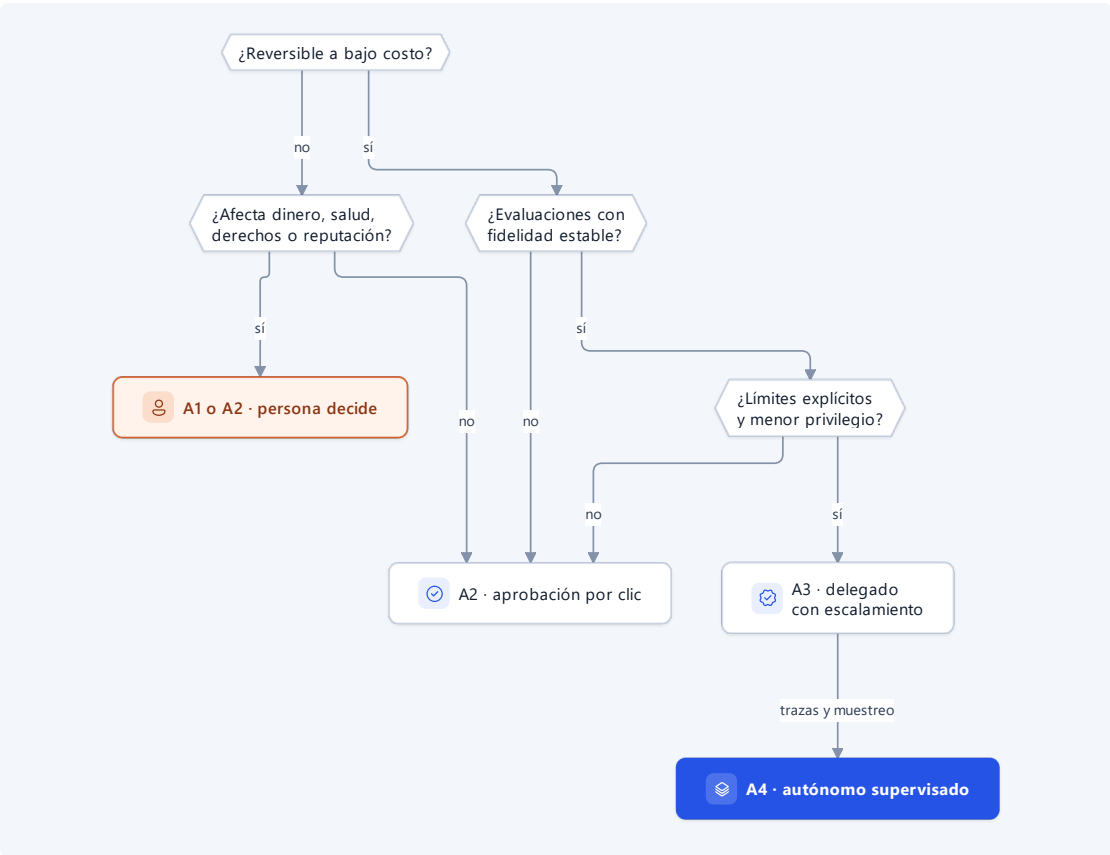


Figura 1.4 — Árbol de decisión para asignar un nivel del Espectro de Autonomía Acotada a cada acción. El paso de A3 a A4 exige trazas, presupuesto y supervisión por muestreo en operación: se sube de nivel por evidencia, nunca por entusiasmo.

Tres reglas de uso. La primera: el nivel se asigna por acción. Un agente de compras puede estar en A3 para cotizar con proveedores homologados y en A1 para recomendar un proveedor nuevo. La segunda: se sube de nivel con evidencia acumulada (semanas de evaluaciones estables, tasa de reversión baja, ausencia de incidentes), y se baja de inmediato ante un incidente. La tercera, la que más cuesta: el nivel de autonomía es un requisito de negocio, no una característica técnica, y lo firma el dueño del proceso junto con el arquitecto. Cuando lo decide sólo el equipo técnico, suele quedar demasiado alto.

Gartner pronosticó en junio de 2025 que más del 40 % de los proyectos de IA agéntica se cancelarán antes de que termine 2027, por costos crecientes, valor de negocio poco claro o controles de riesgo insuficientes. Mi lectura es que buena parte de esas cancelaciones se explican por haber empezado en A4 con la madurez operativa de un A1. El capítulo 6 retoma el espectro para elegir patrones agénticos concretos, y el capítulo 9 lo cruza con el modelo de amenazas.

EN EL STACK MICROSOFT

Los niveles del espectro se mapean bien a piezas concretas. A1 y A2 son el terreno natural de Microsoft 365 Copilot y de los agentes conversacionales de Copilot Studio. A3 y A4 aparecen cuando Copilot Studio usa orquestación generativa con disparadores por evento (agentes autónomos que actúan sin un mensaje del usuario), o cuando se construye en pro-code con Microsoft Agent Framework, sucesor de Semantic Kernel y AutoGen, cuya versión 1.0 para .NET y Python es GA desde abril de 2026, y se despliega en Foundry Agent Service como *hosted agent*, modalidad que ya es GA. Los controles transversales también tienen nombre: Microsoft Entra para la identidad del agente, la política `llm-token-limit` de Azure API Management como AI Gateway para presupuestos de tokens por consumidor, y los evaluadores y el tracing de Microsoft Foundry para la evidencia que justifica subir de nivel. Otras capacidades de Foundry (la memoria del agente, el tracing de agentes externos y de workflows, el evaluador por rúbrica) seguían en preview pública según la tabla de disponibilidad de Foundry consultada en octubre de 2026; conviene verificar su estado antes de comprometerlas en un diseño.

1.7 El arquitecto como diseñador de restricciones

En el software determinista, el arquitecto especifica comportamiento: este servicio recibe esto, valida aquello, persiste lo otro. La especificación es, en el fondo, el programa. Con un componente probabilístico esa estrategia se agota pronto, porque no podemos enumerar el comportamiento correcto para todas las entradas en lenguaje natural. Lo que sí podemos hacer es delimitar el espacio dentro del cual cualquier comportamiento es aceptable. El oficio pasa de dibujar el camino a construir las barandas.

Esas restricciones viven en varias capas, y conviene nombrarlas porque cada una tiene un dueño distinto. Las **restricciones de entrada** deciden qué contexto ve el modelo: qué documentos, filtrados por qué permisos, con qué antigüedad máxima. Las **restricciones de forma** fijan el esquema de salida y los tipos permitidos. Las **restricciones de acción** definen qué herramientas existen, con qué parámetros y con qué identidad se invocan. Las **restricciones de recursos** ponen techo a tokens, pasos, tiempo y dinero por tarea. Y las **restricciones de verificación** establecen qué evaluadores, filtros y revisores humanos se interponen antes de que algo llegue al mundo.

Un ejemplo de lo que pasa un martes a las 3 a.m. ilustra por qué importa. Un agente de soporte con acceso a una herramienta de reembolsos recibe un correo largo que incluye, escondida en una firma, una instrucción para reembolsar el máximo. Si la única defensa es el prompt de sistema ("nunca hagas reembolsos superiores a..."), el resultado depende de una muestra de una distribución. Si la herramienta de reembolsos tiene un límite duro en la API, valida que el caso exista en el CRM y exige aprobación por encima de cierto monto, el peor escenario es un ticket rechazado y una alerta. El modelo es el mismo en ambos diseños; cambia el lugar donde se puso la restricción. (La inyección de prompt indirecta tiene su propio capítulo, el 9; aquí sólo interesa el principio.)

De ahí se desprende una regla práctica para cada revisión: **toda restricción que importe debe vivir fuera del modelo.** El prompt orienta y mejora la tasa de acierto; el código, la API y la política de identidad garantizan. Una regla de negocio que sólo existe como frase en las instrucciones funciona como una esperanza.

El prompt orienta. La arquitectura garantiza. Si una regla sólo existe en las instrucciones del modelo, es una esperanza con buena redacción.

Esto cambia el perfil del arquitecto. A la integración, los datos y la seguridad se suman diseñar experimentos, leer resultados estadísticos, entender la economía del token y aceptar que parte del sistema se valida por muestreo (el capítulo 15 vuelve sobre esto). Una observación incómoda: muchos de los mejores arquitectos que conozco se sienten menos seguros hoy que hace cinco años. Los que se sienten muy seguros suelen estar tratando al modelo como una API más.

ANTIPATRÓN

El "prompt monolítico": un único bloque de instrucciones de varias páginas que mezcla tono, reglas de negocio, límites de seguridad, formato de salida y casos especiales acumulados tras cada incidente. Nadie se atreve a tocarlo, cada cambio rompe algo distinto y es imposible de evaluar por partes. Duele de verdad al migrar de modelo por retiro de versión: algo cambia y nadie sabe qué regla se perdió. La salida es la de cualquier monolito: separar responsabilidades, sacar las reglas duras a código y evaluar cada pieza.

Lo que sigue en el libro desarrolla cada capa de restricción con detalle. El capítulo 2 se ocupa de elegir el componente probabilístico en sí, es decir, qué modelo, de qué tamaño y en qué plataforma. El 3 convierte las ideas de este capítulo en principios de diseño explícitos. Pero la pregunta de fondo ya está planteada, y vale la pena llevarla a la próxima revisión de arquitectura: para cada acción que el sistema puede ejecutar, ¿en qué nivel de autonomía está, quién lo decidió y qué evidencia lo respalda?

IDEAS CLAVE

- El no determinismo de un LLM tiene tres fuentes: muestreo, infraestructura compartida (falta de invariancia de lote) y versión del modelo. **temperature=0** sólo atenúa la primera.
- El fallo de un modelo no lanza excepciones; llega con HTTP 200. Por eso la calidad se gestiona con contratos de comportamiento estadísticos, no sólo con contratos de interfaz.
- Fidelidad, deriva, costo por tarea y autonomía son atributos de calidad de primera clase y deben tener objetivos numéricos en el documento de arquitectura.
- El sistema de razonamiento propone y el sistema de registro dispone: lectura amplia, escritura estrecha mediante APIs con política, identidad y auditoría.
- La autonomía es una decisión de diseño por acción. El Espectro de Autonomía Acotada (A0–A5) asocia a cada nivel los controles mínimos que lo hacen defendible.
- Las restricciones que importan viven fuera del modelo: en la API, en la identidad del agente y en los presupuestos de recursos.

PREGUNTAS PARA SU EQUIPO

1. ¿Cuántas de nuestras soluciones con IA tienen un conjunto dorado de evaluación versionado, y quién lo mantiene?
2. ¿Qué haremos el día que el modelo en el que se apoya nuestro sistema más crítico llegue a su fecha de retiro? ¿Cuánto nos costó la última migración de versión?
3. Para cada acción que ejecutan nuestros agentes, ¿en qué nivel de autonomía está y quién del negocio lo firmó?
4. ¿Qué reglas de negocio existen hoy sólo dentro de un prompt?
5. ¿Conocemos el costo por tarea completada de cada caso de uso, en p50 y p95, y no sólo la factura mensual agregada?

REFERENCIAS

1. He, H. y Thinking Machines Lab. *Defeating Nondeterminism in LLM Inference*. Septiembre de 2025. <https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/>
2. Microsoft Learn. *Learn how to use reproducible output (preview)*. Consultado en septiembre de 2026. <https://learn.microsoft.com/azure/ai-foundry/openai/how-to/reproducible-output>
3. Chen, L., Zaharia, M. y Zou, J. *How Is ChatGPT's Behavior Changing Over Time?* Harvard Data Science Review, 2024 (preprint arXiv:2307.09009, 2023). <https://hdsr.mitpress.mit.edu/pub/95zitmz/release/1>
4. Microsoft Learn. *Microsoft Foundry Models lifecycle and support policy*. Consultado en septiembre de 2026. <https://learn.microsoft.com/azure/foundry/openai/concepts/model-retirements>
5. Microsoft Learn. *AI workloads on Azure y Design principles for AI workloads on Azure (Azure Well-Architected Framework)*. Consultado en septiembre de 2026. <https://learn.microsoft.com/azure/well-architected/ai/get-started>
6. Microsoft Learn. *Structured outputs (Azure OpenAI en Microsoft Foundry)*. Consultado en septiembre de 2026. <https://learn.microsoft.com/azure/foundry/openai/how-to/structured-outputs>
7. ISO/IEC. *ISO/IEC 25010:2023 Systems and software engineering — Product quality model*. 2023. Resumen de cambios en arc42 Quality Model. <https://qualityarc42.org/articles/iso-25010-update-2023>
8. Microsoft Learn. *Built-in evaluators reference (Microsoft Foundry)*. Consultado en septiembre de 2026. <https://learn.microsoft.com/azure/foundry/concepts/built-in-evaluators>
9. Stanford HAI. *AI Index Report 2025*. 2025. <https://hai.stanford.edu/ai-index/2025-ai-index-report>
10. Moore, G. *Systems of Engagement and the Future of Enterprise IT*. AIIIM, 2011. <https://info.aaiim.org/aaiim-blog/newaaiim/2011/01/19/new-geoffrey-moore-white-paper-on-future-of-enterprise-it>
11. Karpathy, A. *Software Is Changing (Again)*, charla en Y Combinator AI Startup School, 17 de junio de 2025. Video en YC Library y resumen en Latent Space. <https://www.ycombinator.com/libraries/MW-andrej-karpathy-software-is-changing-again> · <https://www.latent.space/p/s3>
12. SAE International. *J3016_202104 Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. 2021. https://saemobilus.sae.org/content/j3016_202104

13. Feng, K. J. K., McDonald, D. W. y Zhang, A. X. *Levels of Autonomy for AI Agents*. arXiv:2506.12469, junio de 2025. <https://arxiv.org/abs/2506.12469>
14. Gartner. *Gartner Predicts Over 40% of Agentic AI Projects Will Be Canceled by End of 2027*. Comunicado de prensa, 25 de junio de 2025. <https://www.gartner.com/en/newsroom/press-releases/2025-06-25-gartner-predicts-over-40-percent-of-agentic-ai-projects-will-be-canceled-by-end-of-2027>
15. Microsoft Learn. *Design autonomous agent capabilities y Orchestrate agent behavior with generative AI* (Copilot Studio). Consultado en septiembre de 2026. <https://learn.microsoft.com/microsoft-copilotstudio/guidance/autonomous-agents>
16. Microsoft Agent Framework Blog. *Microsoft Agent Framework Version 1.0*. 3 de abril de 2026. <https://devblogs.microsoft.com/agent-framework/microsoft-agent-framework-version-1-0/>
17. Microsoft Learn. *Limit large language model API token usage* (Azure API Management). Consultado en septiembre de 2026. <https://learn.microsoft.com/azure/api-management/llm-token-limit-policy>
18. Microsoft Learn. *Microsoft Foundry portal general availability overview* (tabla Feature readiness at GA), *What are hosted agents?* y *Rubric evaluators* (preview). Consultado en octubre de 2026. <https://learn.microsoft.com/azure/foundry/concepts/general-availability>

Los modelos cambian cada trimestre. Los criterios para diseñar con ellos, no.

Por primera vez ponemos en el centro de nuestros sistemas un componente que no responde igual dos veces. Este libro reúne los criterios para diseñar con él: cuánta autonomía darle a un agente y con qué evidencia, dónde trazar la frontera de confianza, cómo medir lo que antes solo se probaba y cuánto cuesta de verdad una conversación.

Dieciséis capítulos, dieciséis modelos de decisión propios, más de sesenta diagramas y una arquitectura de referencia completa sobre Microsoft Foundry, Copilot Studio, Agent Framework, Fabric, Entra y Purview, escrita desde la realidad de los equipos latinoamericanos.