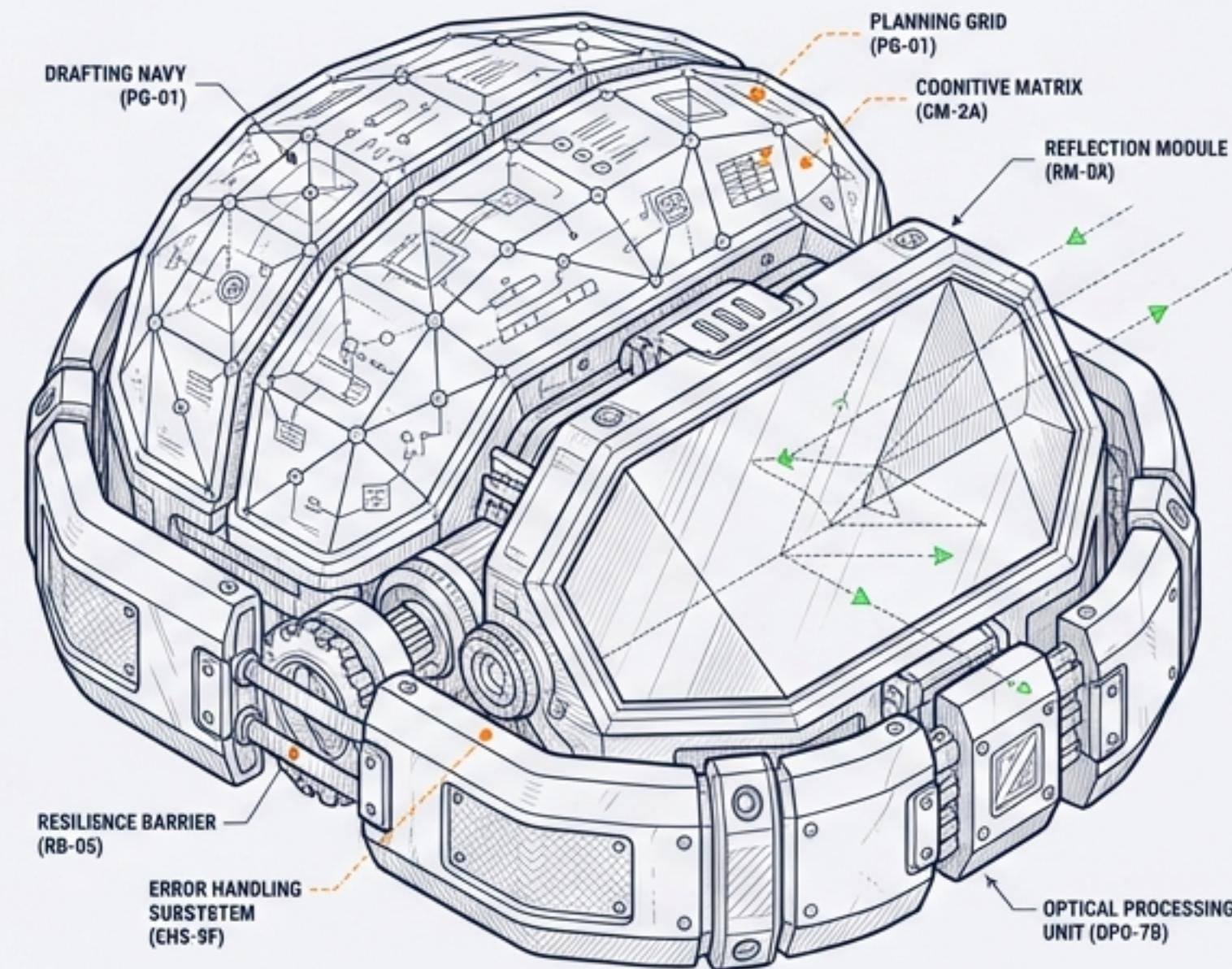


Arquitectura de Agentes Robustos: Planificación, Reflexión y Resiliencia

Patrones avanzados de Prompt Engineering para sistemas de IA autónomos (Módulo 2.3)

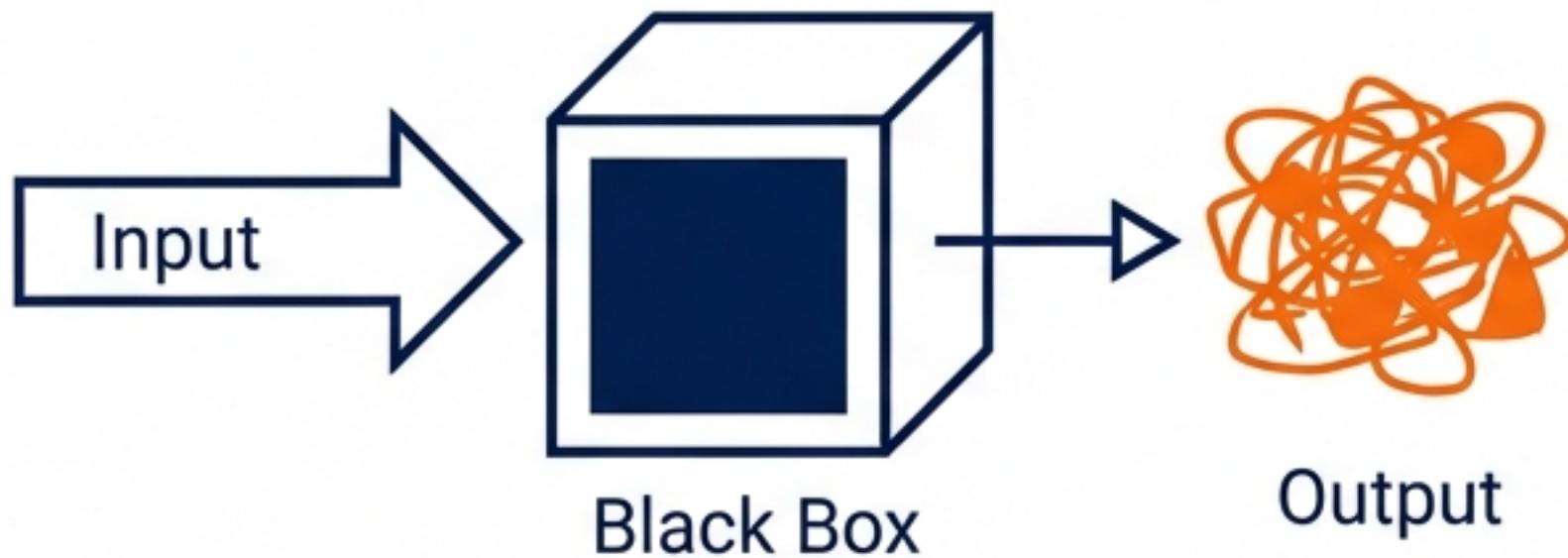


Nivel: Intermedio-Avanzado | Tiempo estimado: 45 min

De la Demo a Producción: Gestionando la Complejidad

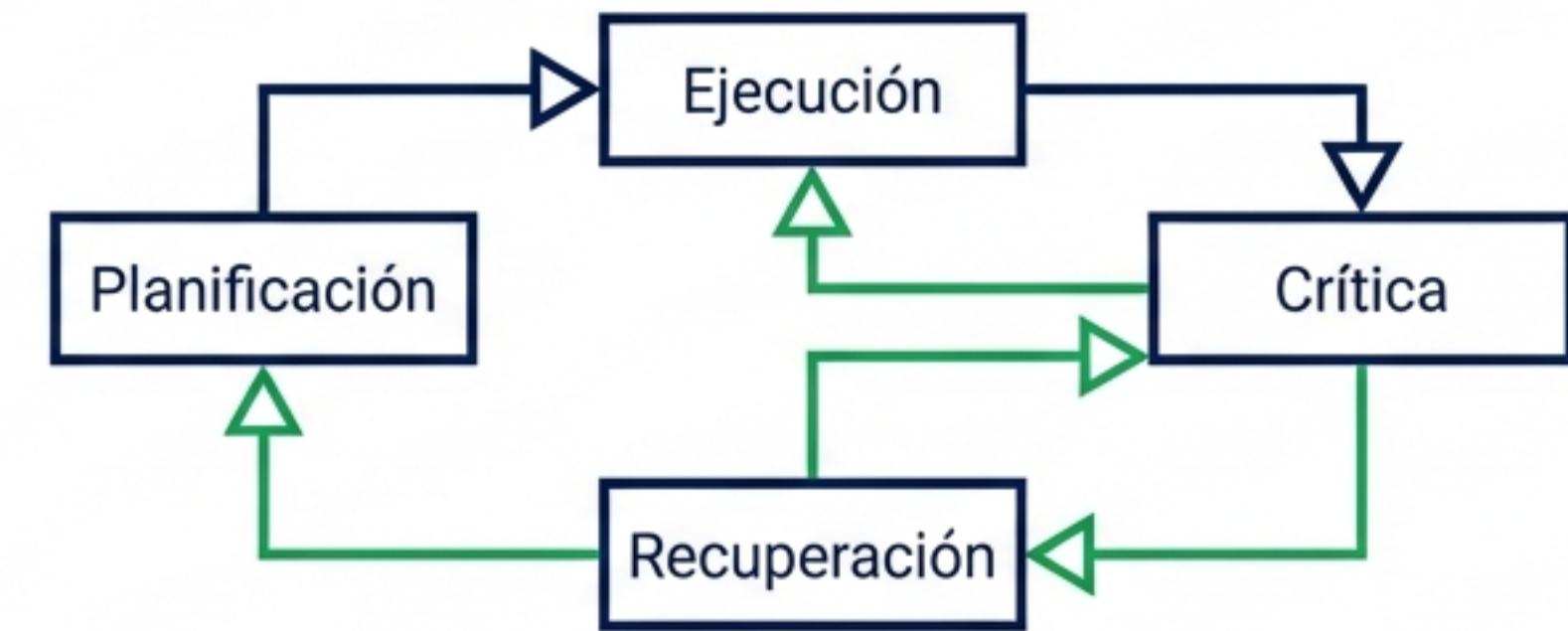
Los LLMs por sí solos son potentes pero impredecibles. Para tareas complejas, la 'fuerza bruta' no escala.

LLM Básico (Fire & Forget)



- Input único
- Caja Negra opaca
- Propenso a alucinaciones
- Lineal y frágil

Agente Robusto (Arquitectura de Sistemas)

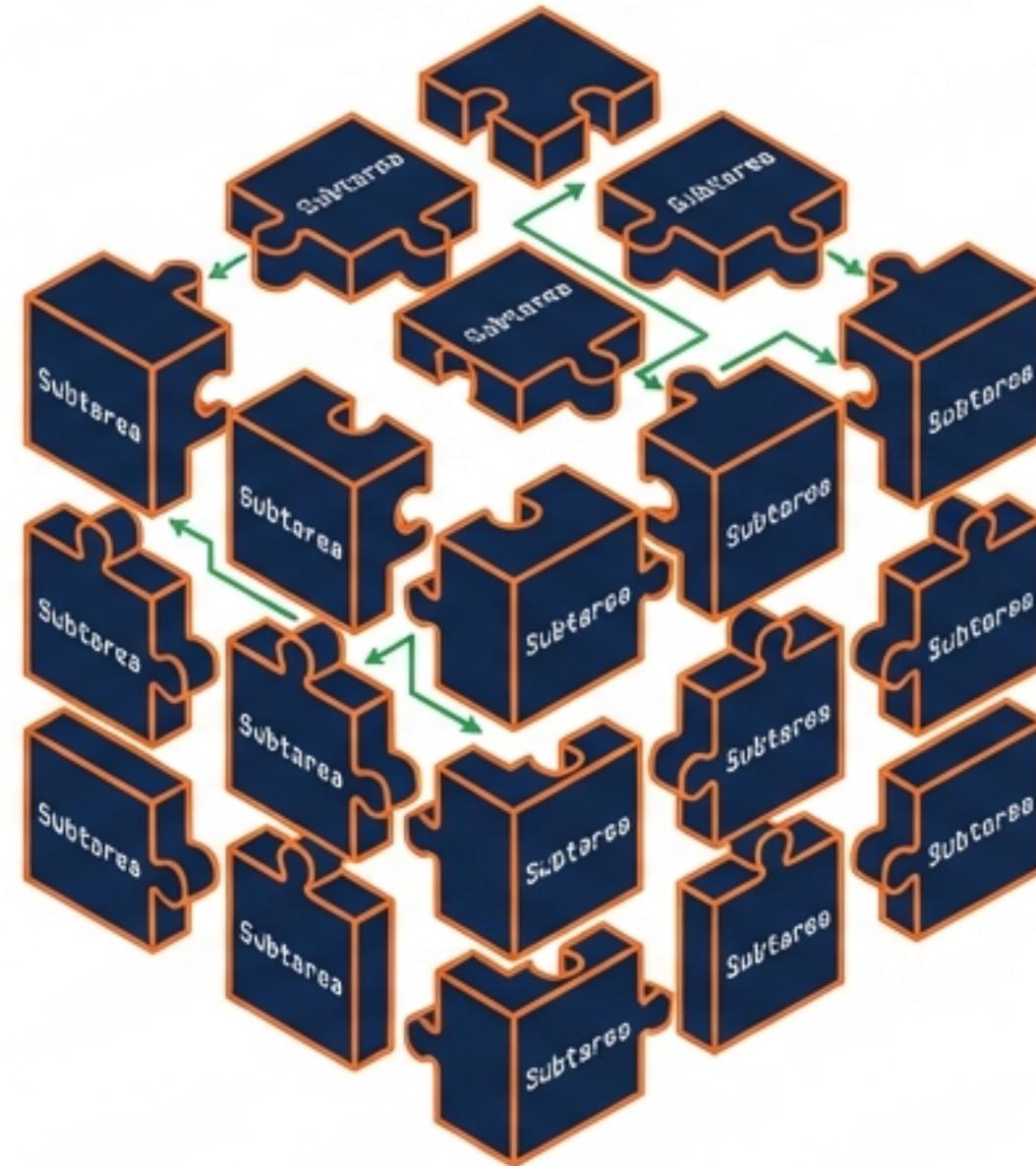


- Planificación estructurada
- Verificación automática
- Recuperación de fallos
- Estable e iterativo

Insight Clave: La diferencia entre un **prototipo** y un **sistema de producción** es la capacidad de **gestionar el fallo** y **verificar la calidad automáticamente**.

Divide y Vencerás: El Poder de la Descomposición (2.3.1)

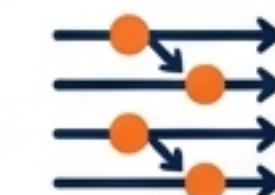
Los agentes efectivos no atacan problemas complejos de un solo golpe.
La descomposición convierte objetivos abstractos en planes ejecutables.



Manejabilidad:
Subtareas digeribles para la ventana de contexto.



Dependencias:
Identificar el orden crítico ($A \rightarrow B$).



Paralelización:
Ejecutar tareas independientes simultáneamente.

Contexto: Frameworks como LangChain y BabyAGI basan su arquitectura en la granularización.

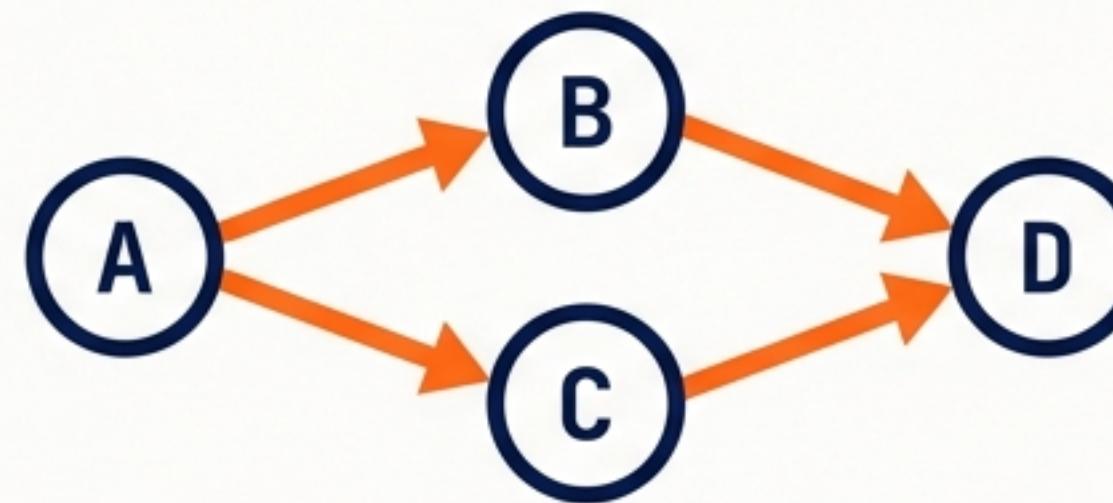
Patrones de Arquitectura de Planificación

1. Secuencial



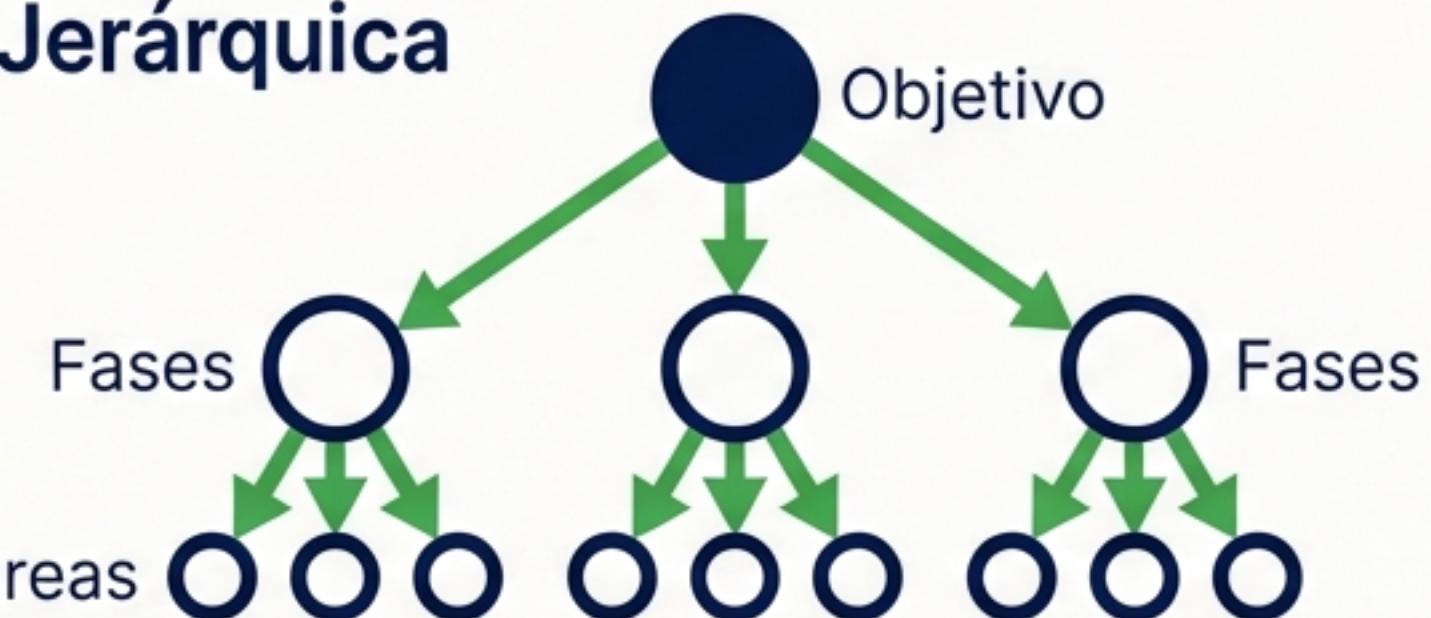
Procesos lineales paso a paso.

3. DAG (Grafo Acíclico Dirigido)



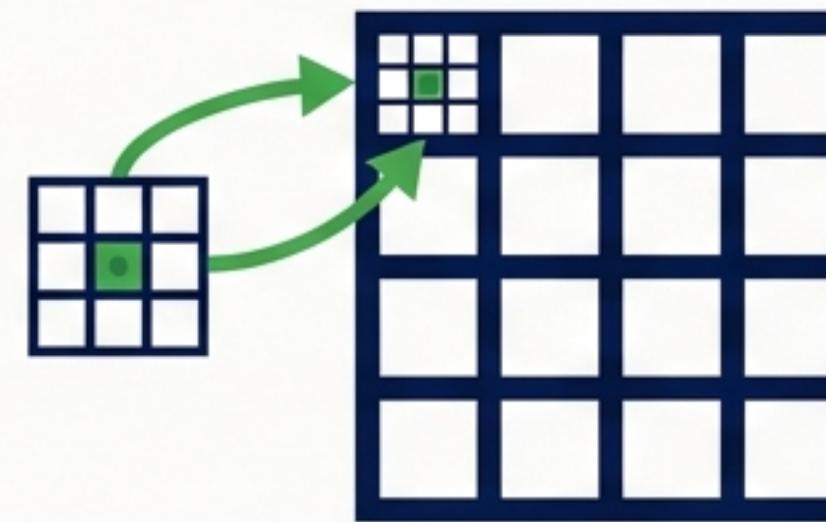
Gestión de dependencias complejas y rutas no lineales.

2. Jerárquica



Niveles de abstracción: Objetivo → Fases → Tareas.

4. Recursiva

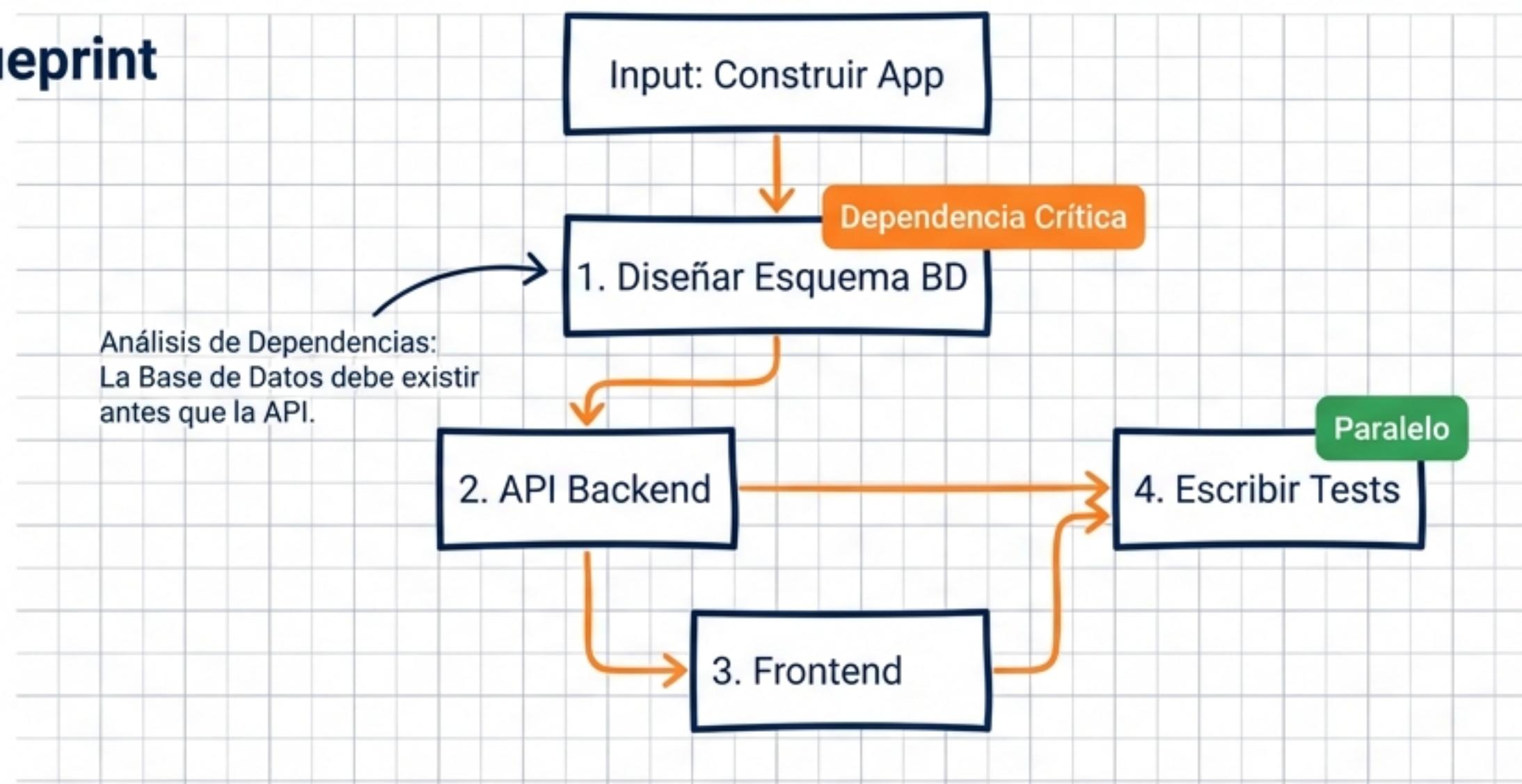


Descomposición hasta nivel atómico (resoluble en un prompt).

Caso Práctico: El Agente de Desarrollo (Fase 1 - Planificación)

Objetivo: “Crear una App de Lista de Tareas (Todo-List) segura”.

Whiteboard Blueprint



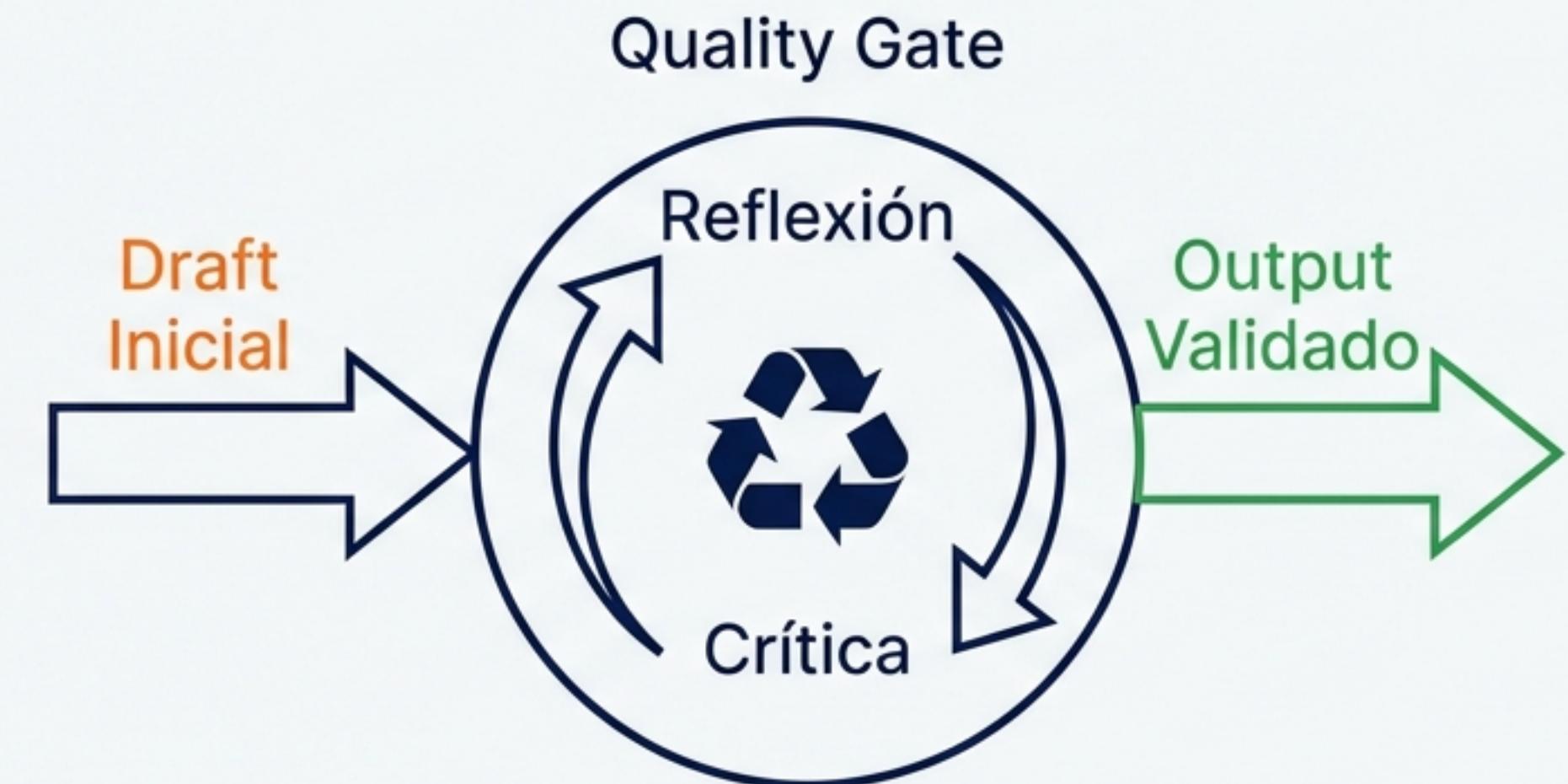
Sin este plan, el agente intentaría generar todo el código en un solo prompt, resultando en alucinaciones y errores de lógica.

La Conciencia del Agente: Reflexión y Auto-Crítica (2.3.2)

La ejecución ciega es peligrosa. La reflexión actúa como un filtro de calidad antes de presentar el resultado final.

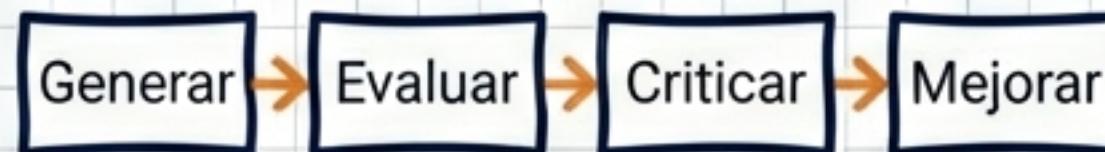
Concepto Clave: Técnica 'Reflexion' (Shinn et al., 2023)

- Ha demostrado mejoras significativas en razonamiento y código.
- Permite detectar alucinaciones y vacíos lógicos.
- Simula un 'Code Review' humano dentro del ciclo de la IA.



Mecánicas de Auto-Mejora

Reflexión Simple



Ciclo básico de mejora iterativa.

Multi-perspectiva



Crítica desde roles asignados específicos.

Memoria de Errores



Consultar historial para no repetir errores previos.

Trade-off: Calidad vs. Costo. Más iteraciones aumentan la precisión pero consumen más tokens y latencia.

Caso Práctico: El Agente de Desarrollo (Fase 2 - Revisión)

El agente revisa el código generado para el Backend.



El Escudo: Manejo de Errores y Resiliencia (2.3.3)

En producción, el 'Happy Path' es un mito.

Las APIs fallan y las respuestas llegan malformadas.

La robustez no es evitar el fallo, es gestionarlo.



- **Recuperación 'Graceful':** Continuar operando a pesar de fallos parciales.
- **Degradación Controlada:** Ofrecer funcionalidad reducida en lugar de una pantalla de error.
- **Limpieza de Estado:** Evitar la 'contaminación de contexto' tras un error.

Estrategias de Recuperación Activa



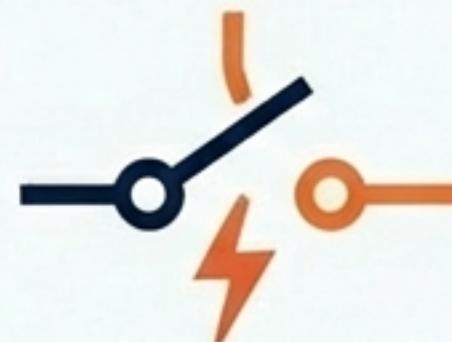
Retry Adaptativo

No solo reintentar. Modificar el prompt basándose en el mensaje de error (ej. corregir sintaxis JSON).



Retry con Backoff

Reintentos con delays exponenciales para no saturar servicios externos.



Circuit Breaker

Mecanismo para detener cascadas de fallos si una herramienta externa está caída.

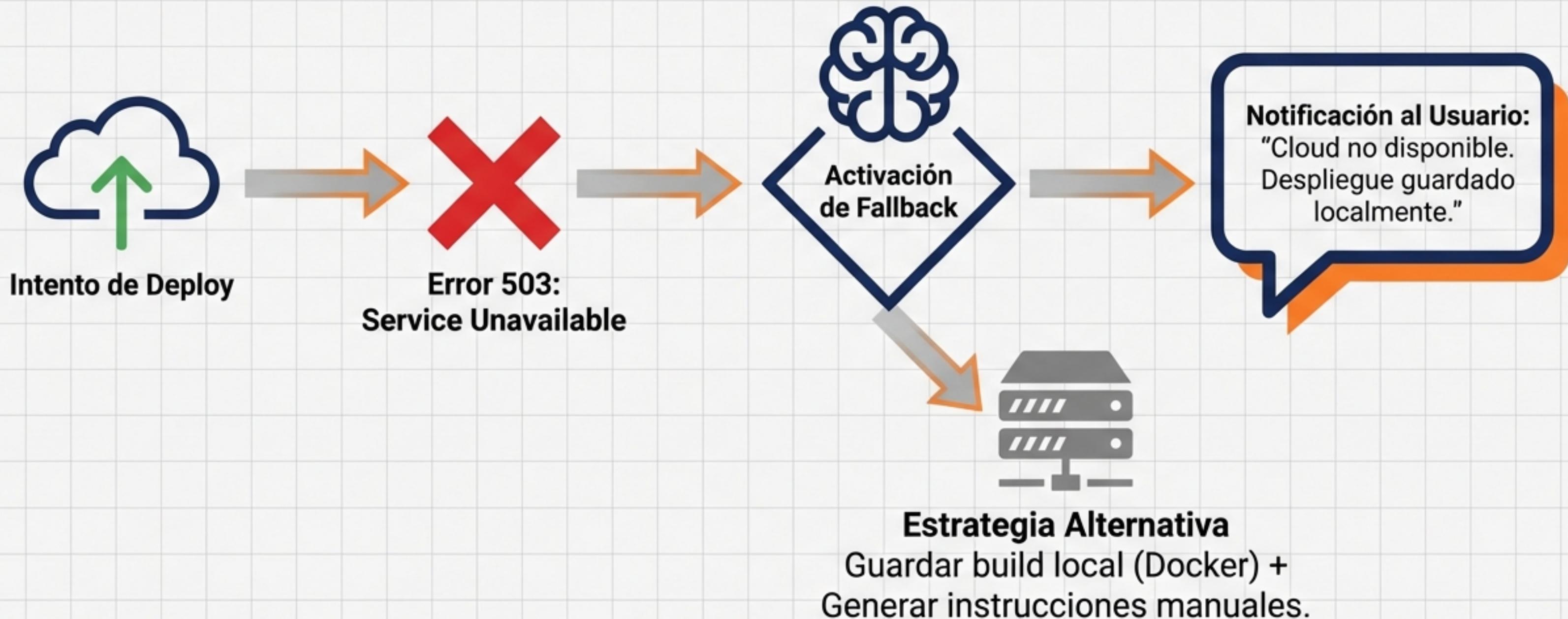


Fallback

Usar una herramienta o método alternativo (Plan B) si el principal falla.

Caso Práctico: El Agente de Desarrollo (Fase 3 - Crisis)

Escenario: Fallo en la API de despliegue en la nube.



Anti-Patrones: Errores Frecuentes de Arquitectura



En Planificación

- Tareas vagas sin criterios de éxito claros.
- Dependencias circulares (A espera a B, B espera a A).



En Reflexión

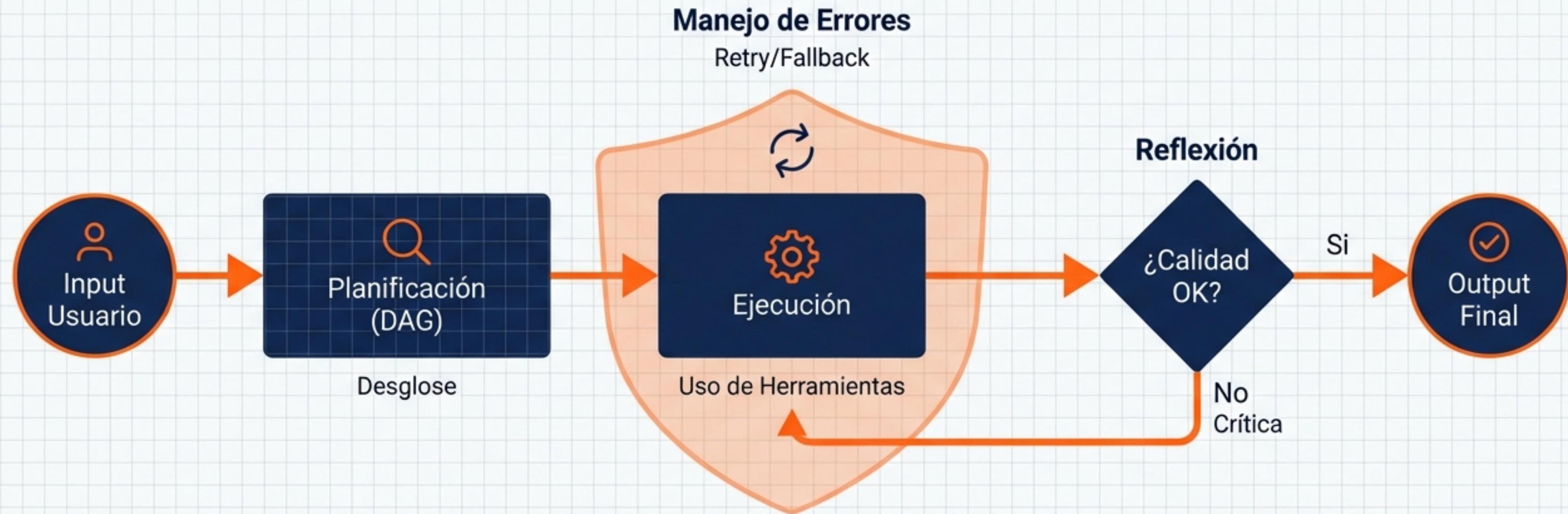
- Bucles infinitos de auto-corrección (parálisis por análisis).
- Auto-crítica destructiva que rompe código funcional.



En Manejo de Errores

- Retries 'ciegos' e infinitos (consumo de presupuesto).
- No limpiar el contexto (el error anterior confunde al siguiente prompt).

El Blueprint Completo: Ciclo de Vida del Agente



Este ciclo convierte la incertidumbre y el caos en resultados fiables.

Principios de Diseño y Próximos Pasos

01. Planificar

Estructurar y descomponer antes de ejecutar.

02. Verificar

La auto-crítica es más barata que corregir un error en producción.

03. Resistir

Diseñar asumiendo que las herramientas fallarán.

Siguiente Módulo

Módulo 3: Function Calling y Herramientas

Ahora que el agente sabe pensar y protegerse, le daremos herramientas para interactuar con el mundo exterior.