Certificación Claude Certified Architect -- Foundations

Guía de estudio (basada en la guía oficial del examen)

Introducción

La certificación **Claude Certified Architect -- Foundations** confirma que el profesional puede tomar decisiones bien fundamentadas sobre compensaciones al implementar soluciones reales basadas en Claude. El examen prueba conocimientos básicos sobre Claude Code, Claude Agent SDK, Claude API y Model Context Protocol (MCP) -- las tecnologías clave para crear aplicaciones de producción con Claude.

Las preguntas del examen se basan en escenarios realistas de la práctica: construcción de sistemas de agentes para soporte al cliente, diseño de pipelines de investigación multiagente, integración de Claude Code en CI/CD, creación de herramientas de productividad para desarrolladores y extracción de datos estructurados de documentos no estructurados.

Candidato objetivo

El candidato ideal es un **arquitecto de soluciones (solution architect)** que diseña e implementa aplicaciones de producción con Claude. Se requiere experiencia de al menos 6 meses con las siguientes tecnologías:

- **Claude Agent SDK** -- orquestación multiagente, delegación a subagentes, integración de herramientas, hooks de ciclo de vida
- **Claude Code** -- CLAUDE.md, servidores MCP, Agent Skills, modo de planificación
- **Model Context Protocol (MCP)** -- herramientas, recursos para integración con backend
- **Ingeniería de prompts** -- esquemas JSON, ejemplos few-shot, plantillas de extracción de datos
- **Ventanas de contexto** -- trabajo con documentos largos, transferencias multiagente
- **Pipelines CI/CD** -- revisión automática de código, generación de pruebas
- **Escalado y confiabilidad** -- manejo de errores, human-in-the-loop

Formato del examen

Parámetro	Valor
Tipo de preguntas	Opción múltiple (1 correcta de 4)
Puntuación	Escala 100-1000, puntuación de aprobación **720**

Penalización por adivinanza	No (¡responde todas las preguntas!)
Escenarios	4 de 8 posibles (seleccionados al azar)

Contenido del examen: 5 dominios

Dominio	Peso
1. Arquitectura de agentes y orquestación	**27%**
2. Diseño de herramientas e integración MCP	**18%**
3. Configuración y flujos de trabajo de Claude Code	**20%**
4. Ingeniería de prompts y salida estructurada	**20%**
5. Gestión de contexto y confiabilidad	**15%**

Escenarios del examen

Escenario 1: Agente de soporte al cliente

Creas un agente para procesar devoluciones, disputas de facturas y problemas de cuenta usando Claude Agent SDK. El agente utiliza herramientas MCP (`get_customer`, `lookup_order`, `process_refund`, `escalate_to_human`). Objetivo: resolución de 80%+ en el primer contacto con escalada adecuada.

Escenario 2: Generación de código con Claude Code

Utilizas Claude Code para acelerar el desarrollo: generación de código, refactorización, depuración, documentación. Necesitas integrarlo con comandos slash personalizados, configuraciones CLAUDE.md y entender cuándo usar el modo de planificación.

Escenario 3: Sistema de investigación multiagente

Un agente coordinador delega tareas a subagentes especializados: búsqueda en internet, análisis de documentos, síntesis y generación de reportes. El sistema debe generar reportes completos con citas.

Escenario 4: Herramientas de productividad para desarrolladores

El agente ayuda a los ingenieros a explorar bases de código desconocidas, generar código boilerplate y automatizar tareas rutinarias. Se utilizan herramientas integradas (Read, Write, Bash, Grep, Glob) y servidores MCP.

Escenario 5: Claude Code para integración continua

Integración de Claude Code en pipelines CI/CD para revisión automática de código, generación de pruebas y retroalimentación en pull requests. Necesitas diseñar prompts con mínimos falsos positivos.

Escenario 6: Extracción de datos estructurados

El sistema extrae información de documentos no estructurados, valida el resultado usando esquemas JSON y mantiene alta precisión. Debe manejar correctamente casos límite.

Escenario 7: Patrones de arquitectura de IA conversacional

Diseñas sistemas conversacionales de múltiples turnos que cubren gestión de ventana de contexto, persistencia de instrucciones a lo largo de los turnos, estrategias de memoria, diseño de herramientas para ejecución segura y manejo de entradas de usuario ambiguas o contradictorias.

Escenario 8: Herramientas de IA agéntica *(contenido faltante — ¡ayúdanos a completarlo!)*

Este escenario ha sido reportado por candidatos del examen pero aún no está cubierto en esta guía. Si has encontrado preguntas de este escenario en el examen real, compártelas en GitHub Issues para que podamos añadir cobertura completa. Tu contribución ayudará a todos los que se preparan para el examen.

Documentación oficial

Recurso	URL
Claude API -- Messages	https://platform.claude.com/docs/en/api/messages
Claude API -- Tool Use	https://platform.claude.com/docs/en/build-with-claude/tool-use
Claude API -- Message Batches	https://platform.claude.com/docs/en/build-with-claude/message-batches
Claude Agent SDK -- Descripción general	https://platform.claude.com/docs/en/agent-sdk/overview
Claude Agent SDK -- Hooks	https://platform.claude.com/docs/en/agent-sdk/hooks
Claude Agent SDK -- Subagentes	https://platform.claude.com/docs/en/agent-sdk/subagents
Claude Agent SDK -- Sesiones	https://platform.claude.com/docs/en/agent-sdk/sessions
Model Context Protocol (MCP)	https://modelcontextprotocol.io/
MCP -- Herramientas	https://modelcontextprotocol.io/docs/concepts/tools
MCP -- Recursos	https://modelcontextprotocol.io/docs/concepts/resources
MCP -- Servidores	https://modelcontextprotocol.io/docs/concepts/servers
Claude Code -- Documentación	https://code.claude.com/docs/en/overview
Claude Code -- CLAUDE.md y memoria	https://code.claude.com/docs/en/memory

Claude Code -- Skills (incluyendo comandos slash)	https://code.claude.com/docs/en/skills
Claude Code -- Hooks	https://code.claude.com/docs/en/hooks
Claude Code -- Subagentes	https://code.claude.com/docs/en/sub-agents
Claude Code -- Integración MCP	https://code.claude.com/docs/en/mcp
Claude Code -- GitHub Actions CI/CD	https://code.claude.com/docs/en/github-actions
Claude Code -- GitLab CI/CD	https://code.claude.com/docs/en/gitlab-ci-cd
Claude Code -- Modo sin interfaz (no interactivo)	https://code.claude.com/docs/en/headless
Guía de ingeniería de prompts	https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/overview
Extended Thinking	https://platform.claude.com/docs/en/build-with-claude/extended-thinking
Anthropic Cookbook (ejemplos de código)	https://github.com/anthropics/anthropic-cookbook

---

PARTE I: BASE TEÓRICA

En esta sección se explica toda la teoría necesaria para aprobar el examen. El material está organizado por tecnologías y conceptos, no por dominios del examen -- esto permite una comprensión más profunda de cada tema.

Capítulo 1: Claude API -- fundamentos de interacción con el modelo

> Documentación: Messages API | Ingeniería de prompts

1.1 Estructura de una solicitud API

Claude API funciona en el principio "solicitud-respuesta". Cada solicitud a Claude Messages API contiene:

```
{
  "model": "claude-sonnet-4-6",
  "max_tokens": 1024,
  "system": "Eres un asistente útil.",
  "messages": [
    {"role": "user", "content": "¡Hola!"},
    {"role": "assistant", "content": "¡Hola!"},
    {"role": "user", "content": "¿Cómo estás?"}
  ],
  "tools": [...],
  "tool_choice": {"type": "auto"}
}
```

Campos clave:

- `model` -- selección del modelo (claude-opus-4-6, claude-sonnet-4-6, claude-haiku-4-5)
- `max_tokens` -- número máximo de tokens en la respuesta
- `system` -- prompt del sistema (establece el comportamiento del modelo)
- `messages` -- historial de conversación (es obligatorio pasar el historial completo para mantener la coherencia)
- `tools` -- definiciones de herramientas disponibles
- `tool_choice` -- estrategia de selección de herramientas

1.2 Roles de los mensajes

El array `messages` utiliza tres roles:

- `user` -- mensajes del usuario
- `assistant` -- respuestas del modelo (se incluyen al pasar el historial)
- `tool` -- resultados de las llamadas a herramientas (el rol no se especifica explícitamente, es un bloque de contenido `tool_result`)

Crítico: en cada solicitud a la API, debes pasar el **historial completo de conversación**. El modelo no mantiene estado entre solicitudes -- cada llamada es independiente.

1.3 El campo stop_reason en la respuesta

La respuesta de Claude API contiene un campo `stop_reason` que determina por qué el modelo dejó de generar:

Valor	Descripción	Acción
`"end_turn"`	El modelo completó su respuesta	Mostrar resultado al usuario
`"tool_use"`	El modelo quiere llamar una herramienta	Ejecutar la herramienta, devolver el resultado

`"max_tokens"`	Se alcanzó el límite de tokens	La respuesta está truncada, puede ser necesario aumentar el límite
`"stop_sequence"`	Se encontró una secuencia de parada	Procesar según la lógica

Para los sistemas de agentes, los más importantes son `"tool_use"` y `"end_turn"` -- controlan el ciclo del agente.

1.4 Prompt del sistema (system prompt)

El prompt del sistema es un mensaje especial que establece el contexto y las reglas de comportamiento del modelo. Tiene estas características:

- No es parte del array `messages`, sino que se transmite en el campo `system`
- Tiene prioridad sobre los mensajes del usuario
- Se carga una vez y actúa durante toda la conversación
- Se utiliza para establecer el rol, limitaciones, formato de salida

Importante para el examen: las formulaciones en el prompt del sistema pueden crear asociaciones no intencionadas con las herramientas. Por ejemplo, la instrucción "siempre verifica el cliente" puede hacer que el modelo llame a `get_customer` demasiado frecuentemente, incluso cuando no es necesario.

1.5 Ventana de contexto

La ventana de contexto es el volumen total de texto (en tokens) que el modelo puede procesar simultáneamente. Incluye:

- El prompt del sistema
- Todo el historial de mensajes
- Definiciones de herramientas
- Resultados de llamadas a herramientas

Problemas clave de la ventana de contexto:

1. **Efecto "pérdida en el medio" (lost-in-the-middle):** los modelos procesan confiablemente la información al principio y al final de una entrada larga, pero pueden perder datos del medio. Solución -- coloca la información clave al principio o al final.
2. **Acumulación de resultados de herramientas:** cada llamada a una herramienta agrega su resultado al contexto. Si una herramienta devuelve 40+ campos pero solo 5 son relevantes -- se desperdicia el 87% del contexto.
3. **Sumarización progresiva:** al condensar el historial, se pierden valores numéricos exactos, porcentajes y fechas, convirtiéndose en expresiones vagas como "aproximadamente", "alrededor de", "algunos".

Capítulo 2: Herramientas (Tools) y tool_use

> Documentación: Tool Use

2.1 ¿Qué es tool_use?

`tool_use` es un mecanismo que permite a Claude llamar funciones externas. El modelo no ejecuta código directamente -- genera una solicitud estructurada para llamar una herramienta, y tu código la ejecuta y devuelve el resultado.

2.2 Definición de una herramienta

Cada herramienta se define mediante un esquema JSON:

```
{
  "name": "get_customer",
  "description": "Busca un cliente por email o ID. Devuelve el perfil del cliente,
incluyendo nombre, email, historial de pedidos y estado de cuenta. Usa esta
herramienta ANTES de lookup_order para verificar la identidad del cliente. Acepta
email (formato: user@domain.com) o customer_id numérico.",
  "input_schema": {
    "type": "object",
    "properties": {
      "email": {"type": "string", "description": "Email del cliente"},
      "customer_id": {"type": "integer", "description": "ID numérico del cliente"}
    },
    "required": []
  }
}
```

Aspectos críticos de la descripción de la herramienta:

1. **La descripción es el mecanismo principal de selección.** El LLM elige la herramienta según su descripción. Descripciones mínimas ("Retrieves customer information") llevan a selecciones erróneas entre herramientas similares.
2. **Incluye en la descripción:**
   - Qué exactamente hace la herramienta y qué devuelve
   - Formatos de entrada y ejemplos de valores
   - Casos límite y limitaciones
   - Cuándo usar esta herramienta vs alternativas similares
3. **Evita:** descripciones idénticas o superpuestas para diferentes herramientas. Si `analyze_content` y `analyze_document` tienen descripciones casi idénticas -- el modelo se confundirá.

4. **Preferencia de herramientas integradas sobre MCP:** Los agentes pueden preferir herramientas integradas (Read, Grep) en lugar de herramientas MCP con funcionalidad similar. Para evitar esto, refuerza las descripciones de herramientas MCP -- especifica ventajas concretas, datos únicos o contexto que las herramientas integradas no proporcionan.

2.3 Parámetro tool_choice

`tool_choice` controla cómo el modelo selecciona herramientas:

Valor	Comportamiento	Cuándo usar
`{"type": "auto"}`	El modelo decide: llamar una herramienta o responder con texto	Por defecto, para la mayoría de casos
`{"type": "any"}`	El modelo **debe** llamar alguna herramienta	Cuando necesitas salida estructurada garantizada
`{"type": "tool", "name": "extract_metadata"}`	El modelo **debe** llamar una herramienta específica	Para forzar un orden de ejecución

Escenarios importantes:

- `tool_choice: "any"` + varias herramientas de extracción -- el modelo selecciona la apropiada, pero garantiza devolver datos estructurados
- Selección forzada -- cuando necesitas asegurar un primer paso específico (por ejemplo, primero `extract_metadata`, luego enriquecimiento)

2.4 Esquemas JSON para salida estructurada

Usar `tool_use` con esquemas JSON es la forma **más confiable** de obtener salida estructurada de Claude. Esto:

- Garantiza la corrección sintáctica del JSON (sin llaves sin cerrar, comas adicionales)
- Asegura conformidad con la estructura (campos requeridos presentes)
- **NO garantiza** corrección semántica (los valores pueden ser incorrectos)

Diseño de esquema -- principios clave:

```
{
  "type": "object",
  "properties": {
    "category": {
      "type": "string",
      "enum": ["bug", "feature", "docs", "unclear", "other"]
    },
    "category_detail": {
      "type": ["string", "null"],
      "description": "Detalles si category = 'other' u 'unclear'"
    },
    "severity": {
      "type": "string",
      "enum": ["critical", "high", "medium", "low"]
    },
    "confidence": {
      "type": "number",
      "minimum": 0,
      "maximum": 1
    },
    "optional_field": {
      "type": ["string", "null"],
      "description": "Null si la información no se encuentra en la fuente"
    }
  },
  "required": ["category", "severity"]
}
```

Reglas de diseño de esquemas:

1. **Required vs Optional:** marca campos como required solo si la información siempre está disponible. Los campos obligatorios fuerzan al modelo a inventar valores si no están en la fuente.
2. **Campos nullables:** usa `"type": ["string", "null"]` para información que puede no estar presente. El modelo devolverá `null` en lugar de inventar.
3. **Enum con "other":** para categorización, agrega `"other"` + detail string para no perder datos fuera de categorías predefinidas.
4. **Enum "unclear":** para casos donde el modelo no puede determinar la categoría con precisión -- un "unclear" honesto es mejor que una categoría errónea.

2.5 Distinción entre errores sintácticos y semánticos

Tipo de error	Ejemplo	Solución
Sintáctico	JSON inválido, tipo de campo incorrecto	tool_use con esquema JSON (lo elimina completamente)
Semántico	Los totales de las filas no coinciden, valor en campo incorrecto, fabricación	Validaciones, reintento con retroalimentación, auto-corrección

Capítulo 3: Claude Agent SDK -- construcción de sistemas de agentes

> Documentación: Agent SDK | Hooks | Subagents | Sessions

3.1 ¿Qué es el ciclo de agente (Agentic Loop)?

El ciclo de agente es el patrón principal para ejecutar tareas de forma autónoma. El modelo no solo responde a una pregunta, sino que ejecuta una secuencia de acciones:

```
1. Enviar solicitud a Claude con herramientas
2. Obtener respuesta
3. Verificar stop_reason:
   - "tool_use" -> ejecutar herramienta, agregar resultado al historial, ir al paso 1
   - "end_turn" -> tarea completada, mostrar resultado al usuario
4. Repetir hasta finalización
```

Este es un enfoque impulsado por el modelo (model-driven): Claude decide qué herramienta llamar a continuación basándose en el contexto y los resultados de acciones anteriores. Esto lo diferencia de los árboles de decisión predefinidos, donde la secuencia de acciones es rígida.

Antipatrones (qué evitar):

- Analizar el texto del asistente para determinar finalización ("Tarea completada" en el texto de respuesta)
- Establecer un límite arbitrario de iteraciones (max_iterations=5) como mecanismo de parada principal
- Verificar el contenido de texto de la respuesta como indicador de finalización

Enfoque correcto: la única señal confiable de finalización es `stop_reason == "end_turn"`.

3.2 Configuración de AgentDefinition

`AgentDefinition` es el objeto de configuración del agente en Claude Agent SDK:

```python
agent = AgentDefinition(
    name="customer_support",
    description="Procesa solicitudes de clientes sobre devoluciones y problemas de pedidos",
    system_prompt="Eres un agente de soporte al cliente...",
    allowed_tools=["get_customer", "lookup_order", "process_refund", "escalate_to_human"],
    # Para el coordinador:
    # allowed_tools=["Task", "get_customer", ...]
)
```

Parámetros clave:

- `name` / `description` -- identificación y descripción del agente
- `system_prompt` -- prompt del sistema con instrucciones
- `allowed_tools` -- lista de herramientas permitidas (principio de privilegios mínimos)

3.3 Hub-and-spoke: coordinador y subagentes

La arquitectura multiagente se construye con el principio "hub-and-spoke" (topología en estrella):

```
        Coordinador
       /     |      \
 Subagente1  Subagente2  Subagente3
 (búsqueda) (análisis)    (síntesis)
```

El coordinador es responsable de:

- Descomposición de la tarea en subtareas
- Decidir qué subagentes se necesitan (selección dinámica, no siempre todo el pipeline)
- Delegar tareas a subagentes
- Agregar y validar resultados
- Manejo de errores y reintentos
- Comunicación de resultados al usuario

Principio crítico: los subagentes tienen contexto aislado.

- Los subagentes **NO heredan** automáticamente el historial de conversación del coordinador
- Todo el contexto debe ser **pasado explícitamente** en el prompt del subagente
- Los subagentes no comparten memoria entre llamadas
- Toda comunicación pasa por el coordinador (para observabilidad y control de errores)

3.4 Herramienta Task para generar subagentes

Los subagentes se generan a través de la herramienta `Task`:

```
# allowed_tools del coordinador debe incluir "Task"
coordinator_agent = AgentDefinition(
    allowed_tools=["Task", "get_customer"]
)
```

Pasar contexto explícitamente es obligatorio:

```
# Malo: el subagente no conoce el contexto
Task: "Analiza el documento"

# Bueno: contexto completo en el prompt
Task: "Analiza el siguiente documento.
Documento: [texto completo del documento]
Resultados de búsqueda anterior: [resultados de búsqueda web]
Requisitos de formato de salida: [esquema]"
```

Generación paralela: el coordinador puede llamar a varios `Task` en una respuesta -- los subagentes se iniciarán en paralelo:

```
# Una respuesta del coordinador contiene:
Task 1: "Buscar artículos sobre tema X"
Task 2: "Analizar documento Y"
Task 3: "Buscar artículos sobre tema Z"
# Los tres se iniciarán simultáneamente
```

3.5 Hooks (Ganchos) en Agent SDK

Los hooks son un mecanismo para interceptar y transformar en puntos específicos del ciclo de vida del agente.

PostToolUse -- intercepta el resultado de la herramienta antes de pasarlo al modelo:

```python
# Ejemplo: normalizar formatos de fecha de diferentes herramientas MCP
@hook("PostToolUse")
def normalize_dates(tool_result):
    # Convertir timestamp Unix -> ISO 8601
    # Convertir "Mar 5, 2025" -> "2025-03-05"
    return normalized_result
```

Hook de intercepción de llamadas salientes -- bloquea acciones que violen política:

```python
# Ejemplo: bloquear reembolsos mayores a $500
@hook("PreToolUse")
def enforce_refund_limit(tool_call):
    if tool_call.name == "process_refund" and tool_call.args.amount > 500:
        return redirect_to_escalation(tool_call)
```

Distinción clave: hooks vs instrucciones en prompts

Característica	Hooks	Instrucciones en prompts
Garantía	**Determinística** (100%)	**Probabilística** (>90%, pero no 100%)

Cuándo usar	Reglas de negocio críticas, operaciones financieras, cumplimiento	Preferencias generales, recomendaciones, formato
Ejemplo	Bloquear reembolsos >$500	"Intenta resolver el problema antes de escalar"

Regla: cuando un error tiene consecuencias financieras, legales o de seguridad -- usa hooks, no prompts.

Capítulo 4: Model Context Protocol (MCP)

> Documentación: MCP | Tools | Resources | Servers

4.1 ¿Qué es MCP?

Model Context Protocol (MCP) es un protocolo abierto para conectar sistemas externos a Claude. MCP define tres tipos principales de recursos:

1. **Tools (herramientas)** -- funciones que el agente puede llamar para ejecutar acciones (operaciones CRUD, llamadas a API, ejecución de comandos)
2. **Resources (recursos)** -- datos a los que el agente puede acceder para obtener contexto (documentación, esquemas de BD, catálogos de contenido)
3. **Prompts (prompts)** -- plantillas de prompts predefinidas para tareas típicas

4.2 Servidores MCP

Un servidor MCP es un proceso que implementa el protocolo MCP y proporciona herramientas/recursos. Al conectarse a un servidor MCP:

- Todas las herramientas se descubren automáticamente
- Las herramientas de todos los servidores conectados están disponibles simultáneamente
- Las descripciones de herramientas determinan cómo el modelo las usará

4.3 Configuración de servidores MCP

Configuración de proyecto (`.mcp.json`) -- para uso en equipo:

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${GITHUB_TOKEN}"
      }
    },
    "jira": {
      "command": "npx",
      "args": ["-y", "mcp-server-jira"],
      "env": {
        "JIRA_TOKEN": "${JIRA_TOKEN}"
      }
    }
  }
}
```

Aspectos clave:

- El archivo `.mcp.json` se almacena en la raíz del proyecto y se gestiona mediante VCS
- Las variables de entorno (`${GITHUB_TOKEN}`) se usan para secretos -- los tokens no se hacen commit
- Disponible para todos los miembros del equipo

Configuración de usuario (`~/.claude.json`) -- para servidores personales/experimentales:

- Se almacena en el directorio de inicio del usuario
- No se transmite mediante VCS
- Adecuado para experimentos personales y pruebas

Selección de servidores:

- Para integraciones estándar (Jira, GitHub, Slack) -- prefiere servidores MCP comunitarios existentes
- Servidores propios -- solo para flujos de trabajo únicos específicos del equipo

4.4 Bandera isError en MCP

Cuando hay un error en una herramienta MCP, se usa la bandera `isError: true` en la respuesta. Esto señala al agente que la llamada falló.

Error estructurado (correcto):

```
{
  "isError": true,
  "content": {
    "errorCategory": "transient",
    "isRetryable": true,
    "message": "El servicio no está disponible temporalmente. Tiempo de espera al
llamar a la API de pedidos.",
    "attempted_query": "order_id=12345",
    "partial_results": null
  }
}
```

Error genérico (antipatrón):

```
{
  "isError": true,
  "content": "Operation failed"
}
```

Un error genérico no proporciona información al agente para tomar una decisión -- ¿reintentar? ¿cambiar la solicitud? ¿escalar?

4.5 Recursos MCP (Resources)

Los recursos son datos que el agente puede solicitar para obtener contexto sin ejecutar acciones:

- Catálogos de contenido (lista de todas las tareas en el proyecto, navegación de jerarquía)
- Esquemas de bases de datos (comprensión de la estructura de datos)
- Documentación (referencia de API, guías internas)
- Resúmenes de problemas/tareas

Ventaja de los recursos: el agente no necesita hacer llamadas de herramienta exploratoria para entender los datos disponibles. El recurso proporciona un "mapa" inmediatamente.

Capítulo 5: Claude Code -- configuración y flujos de trabajo

> Documentación: Claude Code | Memory / CLAUDE.md | Skills | MCP | Hooks | Sub-agents | GitHub Actions | Headless

5.1 Jerarquía de CLAUDE.md

CLAUDE.md es archivo(s) con instrucciones para Claude Code. Existe una jerarquía con tres niveles:

```
1. Usuario: ~/.claude/CLAUDE.md
   - Se aplica solo a este usuario
   - NO se transmite mediante VCS
   - Preferencias personales, estilo de trabajo

2. Proyecto: .claude/CLAUDE.md o CLAUDE.md en la raíz
   - Se aplica a todos los miembros del equipo
   - Gestionado mediante VCS
   - Estándares de codificación, pruebas, decisiones arquitectónicas

3. Directorio: CLAUDE.md en subdirectorios
   - Se aplica al trabajar con archivos en este directorio
   - Convenciones específicas para esta parte de la base de código
```

Error típico: un nuevo miembro del equipo no recibe instrucciones de proyecto porque fueron colocadas en `~/.claude/CLAUDE.md` (nivel de usuario) en lugar de `.claude/CLAUDE.md` (nivel de proyecto).

5.2 Sintaxis @path (importación de archivos)

CLAUDE.md puede referenciar archivos externos usando la sintaxis `@path`, haciendo la configuración modular:

```
# CLAUDE.md del proyecto

Los estándares de codificación se describen en @./standards/coding-style.md
Los requisitos de pruebas están en @./standards/testing-requirements.md
La descripción del proyecto está en @README.md y las dependencias en @package.json
```

Reglas de sintaxis `@path`:

- Usa `@` directamente antes de la ruta del archivo (sin espacio)
- Se soportan rutas relativas y absolutas
- Las rutas relativas se resuelven relativas al archivo que las contiene
- La profundidad máxima de importes anidados es 5 niveles

Esto evita duplicación y permite que cada paquete incluya solo estándares relevantes.

5.3 Directorio .claude/rules/

`.claude/rules/` es una alternativa a un CLAUDE.md monolítico para organizar reglas por tema:

```
.claude/rules/
  testing.md          -- convenciones de pruebas
  api-conventions.md  -- convenciones de API
  deployment.md       -- reglas de despliegue
  react-patterns.md   -- patrones de React
```

Característica clave: frontmatter YAML con campo `paths` para carga condicional:

```
---
paths: ["src/api/**/*"]
---

Para archivos de API usa async/await con manejo explícito de errores.
Cada endpoint debe devolver un envoltorio de respuesta estándar.
```

```
---
paths: ["**/*.test.tsx", "**/*.test.ts"]
---

Las pruebas deben usar bloques describe/it.
Usa fabricas de datos en lugar de valores hardcodeados.
No simules la base de datos -- usa BD de prueba.
```

Cómo funciona:

- La regla se carga **solo** cuando Claude Code edita un archivo que coincide con el patrón `paths`
- Esto ahorra contexto y tokens -- reglas irrelevantes no se cargan
- Los patrones glob permiten aplicar convenciones a archivos dispersos en múltiples directorios -- ideal para pruebas dispersas en toda la base de código

Cuándo usar `.claude/rules/` con paths vs CLAUDE.md a nivel de directorio:

- `.claude/rules/` con paths -- cuando las convenciones se aplican a archivos dispersos en muchos directorios (pruebas, migraciones)
- CLAUDE.md a nivel de directorio -- cuando las convenciones están vinculadas a un directorio específico y no se necesitan fuera de él

5.4 Comandos slash personalizados y Skills

> **Nota:** en la versión actual de Claude Code, los comandos personalizados (`.claude/commands/`) están combinados con las habilidades (`.claude/skills/`). Ambos formatos crean comandos invocables mediante `/nombre`. La guía de examen se refiere a `.claude/commands/` -- este formato sigue siendo compatible.

Los comandos slash son plantillas de prompts reutilizables, invocadas mediante `/nombre`:

Formato mediante `.claude/commands/` (heredado, compatible):

```
.claude/commands/
  review.md        -- /review -- revisión de código estándar
  test-gen.md      -- /test-gen -- generación de pruebas
```

Formato mediante `.claude/skills/` (actual):

```
.claude/skills/
  review/SKILL.md  -- /review -- con configuración de frontmatter
  test-gen/SKILL.md
```

Comandos de proyecto (`.claude/commands/` o `.claude/skills/`):

- Se almacenan en VCS, disponibles para todos al clonar el repositorio
- Aseguran uniformidad en los flujos de trabajo del equipo

Comandos de usuario (`~/.claude/commands/` o `~/.claude/skills/`):

- Comandos personales, no transmitidos mediante VCS
- Para flujos de trabajo individuales

5.5 Habilidades (Skills) -- .claude/skills/

Las habilidades son comandos extendidos con configuración mediante frontmatter SKILL.md:

```
---
context: fork
allowed-tools: ["Read", "Grep", "Glob"]
argument-hint: "Ruta del directorio para analizar"
---

Analiza la estructura del código en el directorio especificado.
Genera un informe sobre dependencias y patrones arquitectónicos.
```

Parámetros de frontmatter:

Parámetro	Descripción
`context: fork`	Ejecuta la habilidad en un subagente aislado. La salida verbosa no contamina la sesión principal
`allowed-tools`	Restringe las herramientas disponibles (seguridad -- la habilidad no puede eliminar archivos a menos que se permita)
`argument-hint`	Sugerencia que solicita un parámetro al llamar sin argumentos

Cuándo usar habilidad vs CLAUDE.md:

- **Habilidad** -- invocación bajo demanda para una tarea específica (revisión, análisis, generación)
- **CLAUDE.md** -- estándares y convenciones universales siempre cargados

Habilidades personales (`~/.claude/skills/`):

- Crea variantes personales con nombres diferentes para no afectar a colegas

5.6 Modo de planificación (Plan Mode) vs ejecución directa

Modo de planificación:

- El modelo **solo** investiga y planifica, sin hacer cambios
- Usa Read, Grep, Glob para explorar la base de código
- Resultado -- plan de implementación aprobado por el usuario
- Exploración segura sin efectos secundarios

Cuándo usar modo de planificación:

- Cambios a gran escala (decenas de archivos)
- Múltiples enfoques válidos (microservicios: ¿cómo dividir los límites?)
- Decisiones arquitectónicas (¿qué framework? ¿qué estructura?)
- Base de código desconocida (necesita entender antes de cambiar)
- Migraciones de bibliotecas afectando 45+ archivos

Cuándo usar ejecución directa:

- Correcciones de archivo único con stack trace claro
- Agregar una validación
- Cambios bien comprendidos y no ambiguos

Enfoque combinado:

1. Modo de planificación para exploración y diseño
2. Aprobación del plan por el usuario
3. Ejecución directa para implementar el plan aprobado

Subagente Explore -- subagente especializado para explorar la base de código:

- Aísla la salida verbosa del contexto principal
- Devuelve solo un resumen
- Previene el agotamiento de la ventana de contexto en tareas multifase

5.7 Comando /compact

`/compact` es un comando integrado para comprimir contexto:

- Sumariza el historial anterior, liberando la ventana de contexto
- Se usa en sesiones de investigación largas, cuando el contexto se llena de salida verbosa de herramientas
- Riesgo: se pierden valores numéricos exactos, fechas y detalles específicos durante la sumarización

5.8 Comando /memory

`/memory` es un comando integrado para gestionar memoria entre sesiones:

- Abre el archivo `CLAUDE.md` para edición, permitiendo guardar notas, preferencias y contexto
- La información se persiste entre sesiones y se carga automáticamente al iniciar
- Útil para almacenar: convenciones de proyecto, preferencias del usuario, comandos usados frecuentemente, contexto del trabajo actual
- Alternativa a repetir las mismas instrucciones en cada sesión

5.9 Claude Code CLI para CI/CD

Bandera `-p` (o `--print`):

```
claude -p "Analiza esta solicitud de extracción para problemas de seguridad"
```

- Modo no interactivo: procesa el prompt, emite a stdout, se detiene
- Sin esperar entrada del usuario
- **La única forma correcta** de ejecutar en pipelines de CI/CD

Salida estructurada para CI:

```
claude -p "Revisa este PR" --output-format json --json-schema
'{"type":"object",...}'
```

- `--output-format json` -- salida en formato JSON
- `--json-schema` -- validación de salida según esquema
- El resultado se puede analizar para postear automáticamente comentarios inline en PR

Aislamiento de contexto de sesión: La misma sesión de Claude que **generó** código es menos efectiva para **revisarlo** (el modelo retiene el contexto de razonamiento y es menos propenso a cuestionarse a sí mismo). Usa una **instancia independiente** para revisión.

Prevención de comentarios duplicados: Cuando hagas revisión posterior después de nuevos commits -- incluye los resultados de revisión anterior en el contexto e instruye a Claude para reportar solo problemas **nuevos o no arreglados**.

5.10 fork_session y gestión de sesiones

`--resume <session-name>` -- retomar sesión nombrada:

```
claude --resume investigation-auth-bug
```

- Continúa la conversación anterior con contexto guardado
- Útil para investigaciones largas en múltiples sesiones
- Riesgo: si los archivos cambiaron desde la sesión anterior, los resultados de herramientas pueden estar desactualizados

`fork_session` -- crear una rama independiente desde una base común:

```
Exploración de base de código
         |
    fork_session
    /           \
Enfoque A:      Enfoque B:
Redux          Context API
```

- Ambas ramas heredan el contexto hasta el punto de bifurcación
- Luego se desarrollan independientemente
- Útil para comparar enfoques o probar estrategias

Cuándo iniciar una sesión nueva en lugar de reanudar:

- Los resultados de herramientas están desactualizados (archivos cambiaron)
- Ha pasado mucho tiempo y el contexto se degradó
- Es mejor comenzar con "Aquí hay un resumen breve de lo que descubrimos: ..." que reanudar con datos antiguos

---

Capítulo 6: Ingeniería de Prompts -- técnicas avanzadas

> Documentación: Prompt Engineering | Anthropic Cookbook

6.1 Prompting de pocos ejemplos (Few-shot)

Few-shot prompting es incluir 2-4 ejemplos de entrada/salida en el prompt para demostrar el comportamiento esperado.

Por qué few-shot es más efectivo que descripciones de texto:

- Una descripción de texto "sé más preciso" se interpreta de diferentes maneras
- Un ejemplo muestra inequívocamente el formato esperado y la lógica de decisión
- El modelo generaliza el patrón a nuevos casos (no solo repite ejemplos)

Tipos de ejemplos few-shot y su aplicación:

1. **Ejemplos para escenarios ambiguos:**

```
Solicitud: "Mi pedido está roto"
Acción: Llamar get_customer -> lookup_order -> verificar estado.
Justificación: "roto" podría significar artículo dañado. Necesita verificar detalles
del pedido.

Solicitud: "Dame un gerente"
Acción: Llamar inmediatamente escalate_to_human.
Justificación: El cliente solicita explícitamente una persona. No intentar resolver
solo.
```

2. **Ejemplos para formato de salida:**

```
Ejemplo de hallazgo:
{
  "location": "src/auth/login.ts:42",
  "issue": "SQL injection en parámetro username",
  "severity": "critical",
  "suggested_fix": "Usar consulta parametrizada"
}
```

3. **Ejemplos para distinguir código aceptable de problemático:**

```
// Aceptable (no marcar):
const items = data.filter(x => x.active);

// Problema (marcar):
const items = data.filter(x => x.active == true); // Usa comparación estricta ===
```

4. **Ejemplos para extraer de diferentes formatos de documentos:**

```
Documento con citas inline:
"Como se muestra en la investigación (Smith, 2023), el nivel es 42%."
-> {"value": "42%", "source": "Smith, 2023", "type": "inline_citation"}

Documento con bibliografía:
"El nivel es 42%. [1]"
-> {"value": "42%", "source": "reference_1", "type": "bibliography"}
```

5. **Ejemplos para medidas informales:**

```
Texto: "aproximadamente dos puñados de arroz"
-> {"amount": "~100g", "original_text": "dos puñados", "precision": "approximate"}

Texto: "una pizca de sal"
-> {"amount": "~1g", "original_text": "pizca", "precision": "approximate"}
```

Few-shot es particularmente efectivo para extraer unidades informales y no estándar, donde las reglas de texto no pueden cubrir toda la diversidad de expresiones.

Reglas de normalización de formato en prompts: Cuando uses esquemas JSON estrictos para salida estructurada, agrega reglas de normalización en el prompt:

```
Normalización:
- Fechas: siempre ISO 8601 (YYYY-MM-DD), "ayer" -> calcular fecha absoluta
- Moneda: valor numérico + código de moneda, "cinco dólares" -> {"amount": 5,
"currency": "USD"}
- Porcentajes: decimal, "media" -> 0.5
```

Esto previene errores semánticos cuando el JSON es sintácticamente válido pero los valores son inconsistentes.

6.2 Criterios explícitos vs instrucciones vagas

Malo (vago):

```
Verifica los comentarios del código en busca de precisión.
Sé conservador, reporta solo hallazgos de alta confianza.
```

Bueno (criterios explícitos):

```
Marca un comentario como problemático SOLO si:
1. El comentario describe comportamiento que CONTRADICE el comportamiento real del
código
2. El comentario hace referencia a una función o variable que no existe
3. El comentario TODO/FIXME es para un bug que ya está arreglado en el código

NO marques:
- Comentarios que están solo obsoletos estilísticamente
- Comentarios con imprecisiones menores en el lenguaje
- Ausencia de comentarios (esa es otra categoría)
```

Definición de criterios de severidad con ejemplos:

```
CRITICAL: Bloqueo en tiempo de ejecución para usuarios
  Ejemplo: NullPointerException al procesar pago

HIGH: Vulnerabilidad de seguridad
  Ejemplo: SQL injection, XSS, falta de verificación de permisos

MEDIUM: Error lógico sin efecto inmediato
  Ejemplo: Clasificación incorrecta, error off-by-one

LOW: Calidad del código
  Ejemplo: Duplicación, algoritmo subóptimo para datos pequeños
```

6.3 Prompt Chaining (Encadenamiento de Prompts)

Prompt chaining es dividir una tarea compleja en pasos secuenciales y enfocados:

```
Paso 1: Analizar archivo auth.ts (solo problemas locales)
       -> Resultado: lista de problemas en auth.ts

Paso 2: Analizar archivo database.ts (solo problemas locales)
       -> Resultado: lista de problemas en database.ts

Paso 3: Pasada de integración (dependencias entre archivos)
       -> Resultado: problemas en las interfaces de módulos
```

Por qué es necesario:

- Evita **diluir la atención** -- cuando el modelo recibe demasiados archivos simultáneamente, puede pasar por alto bugs en algunos archivos mientras hace comentarios superficiales en otros
- Asegura **calidad consistente** del análisis para cada archivo
- Permite analizar **separadamente** interacciones entre archivos

Cuándo usar prompt chaining vs descomposición dinámica:

- **Prompt chaining** -- para tareas predecibles y repetitivas (revisión de código, migración de archivos)
- **Descomposición dinámica** -- para exploración abierta, donde subtareas se clarifican durante el proceso

6.4 Patrón "entrevista"

Antes de implementar una solución, Claude hace preguntas aclaratorias al desarrollador:

```
Claude: "Antes de implementar almacenamiento en caché para la API, tengo algunas
preguntas:
1. ¿Qué estrategia de invalidación de caché prefieres -- TTL o basada en eventos?
2. ¿Es aceptable datos desactualizados si el caché no está disponible?
3. ¿Necesitamos caché a nivel de usuario individual o global?
4. ¿Cuál es el volumen de datos esperado para almacenar en caché?"
```

Cuándo es útil:

- Dominio desconocido (fintech, medicina, sistemas legales)
- Tarea con implicaciones no obvias (estrategias de caché, modos de fallo)
- Múltiples enfoques válidos, cuya elección depende del contexto

6.5 Validación y retry-with-feedback

Cuando datos extraídos fallan validación:

```
Paso 1: Extraer datos del documento
Paso 2: Validar (Pydantic, JSON Schema, reglas de negocio)
Paso 3: Si error -- solicitar nuevamente con contexto:
  - Documento original
  - Extracción anterior (errónea)
  - Error específico: "Campo 'total' = 150, pero suma de line_items = 145. Verifica los valores."
```

Cuándo el reintento será efectivo:

- Errores de formato (fecha en formato incorrecto)
- Errores de estructura (campo colocado incorrectamente)
- Discrepancias aritméticas (modelo puede reverificar)

Cuándo el reintento NO ayudará:

- Información ausente en la fuente (no en documento -- reintento no lo agregará)
- Contexto externo (datos en otro documento, no pasado al modelo)

Pydantic como herramienta de validación: Pydantic es una biblioteca de Python para validación de datos basada en esquemas. En el contexto del examen, lo importante es:

- **Validación de estructura:** Pydantic verifica tipos de campos, obligatoriedad, valores permitidos (enum) a nivel de código después de recibir JSON de Claude
- **Validación semántica:** Validadores personalizados verifican lógica de negocio (suma de elementos = total, fecha inicio < fecha fin)
- **Ciclos de validación/reintento:** Al error de validación de Pydantic -- formamos mensaje de error y enviamos a Claude solicitud de reintento con contexto del error
- **Generación de esquemas JSON:** Los modelos de Pydantic pueden generar automáticamente JSON Schema para el parámetro `tool_use`, proporcionando una única fuente de verdad para el esquema

6.6 Autocorrección (Self-correction)

Patrón para detectar contradicciones internas:

```json
{
  "stated_total": "$150.00",
  "calculated_total": "$145.00",
  "conflict_detected": true,
  "line_items": [
    {"name": "Widget A", "price": 75.00},
    {"name": "Widget B", "price": 70.00}
  ]
}
```

El modelo extrae **tanto** el valor indicado **como** el calculado -- si divergen, la bandera `conflict_detected` permite manejar la divergencia.

Capítulo 7: API de Message Batches

> Documentación: Message Batches

7.1 Descripción general

La API de Message Batches permite enviar lotes de solicitudes para procesamiento asincrónico:

Característica	Valor
Ahorro	**50%** del costo de llamadas sincrónicas
Ventana de procesamiento	Hasta **24 horas** (sin garantías SLA de latencia)
Multi-turn tool calling	**No soportado** (una solicitud = una respuesta)
Correlación	Campo `custom_id` para vincular solicitud y respuesta

7.2 Cuándo usar Batch API vs API sincrónica

Tarea	API	Razón
Verificación pre-merge de PR	**Sincrónica**	El desarrollador espera resultado; 24 horas inaceptable
Reporte nocturno de deuda técnica	**Batch**	Resultado necesario por la mañana; ahorro del 50%
Auditoría de seguridad semanal	**Batch**	Sin prisa; ahorro del 50%
Revisión de código interactiva	**Sincrónica**	Necesita respuesta inmediata
Procesar 10,000 documentos	**Batch**	Procesamiento masivo; ahorro significativo

7.3 Trabajar con custom_id

```
{
  "custom_id": "doc-invoice-2024-001",
  "params": {
    "model": "claude-sonnet-4-6",
    "max_tokens": 1024,
    "messages": [{"role": "user", "content": "Extrae datos de: ..."}]
  }
}
```

`custom_id` permite:

- Vincular resultado con documento original

- Si falla -- reenviar **solo** documentos fallidos
- No reprocesar documentos exitosos

7.4 Manejo de fallos en lotes

1. Enviar lote de 100 documentos
2. 95 procesan exitosamente, 5 fallan (context limit exceeded)
3. Identificar los fallidos por `custom_id`
4. Modificar (dividir documentos largos en fragmentos)
5. Reenviar solo los 5 fallidos

7.5 Cálculo de SLA

Si necesitas resultado en 30 horas y Batch API procesa hasta 24 horas:

- Ventana de envío: 30 - 24 = **6 horas**
- Los lotes deben enviarse no más de 24 horas antes de la fecha límite
- Para envíos frecuentes -- dividir en ventanas de 4 horas

Capítulo 8: Estrategias de descomposición de tareas

8.1 Pipelines fijos (Prompt Chaining)

Cada paso se define de antemano:

```
Documento -> Extracción de metadatos -> Extracción de datos -> Validación ->
Enriquecimiento -> Salida final
```

Cuándo usar:

- Estructura predecible de tarea (revisión siempre por la misma plantilla)
- Todos los pasos conocidos de antemano
- Se necesita estabilidad y reproducibilidad

8.2 Descomposición adaptativa dinámica

Las subtareas se generan basándose en resultados intermedios:

```
1. "Agrega pruebas para la base de código heredada"
2. -> Primero: mapeo de estructura (Glob, Grep)
3. -> Descubierto: 3 módulos sin pruebas, 2 con cobertura parcial
4. -> Priorización: comenzar con módulo de pagos (alto riesgo)
5. -> Durante el proceso: descubierta dependencia de API externo
6. -> Adaptación: agregar mock para API externo antes de escribir pruebas
```

Cuándo usar:

- Tareas exploratorias abiertas
- Cuando el volumen completo de trabajo es desconocido de antemano
- Cuando cada paso depende de resultados del anterior

8.3 Revisión de código multi-pasada

Para pull requests con 10+ archivos:

```
Pasada 1 (por archivo): Analizar auth.ts -> lista de problemas locales
Pasada 1 (por archivo): Analizar database.ts -> lista de problemas locales
Pasada 1 (por archivo): Analizar routes.ts -> lista de problemas locales
...
Pasada 2 (integración): Analizar vínculos entre archivos
  -> Problemas entre archivos: tipos inconsistentes, dependencias cíclicas
```

Por qué una pasada para 14 archivos es malo:

- Dilución de atención: análisis detallado de algunos archivos, superficial de otros
- Comentarios contradictorios: patrón marcado como problema en un archivo, aprobado en otro
- Bugs perdidos: errores obvios pasados por alto debido a sobrecarga cognitiva

---

Capítulo 9: Escalada y Human-in-the-Loop

9.1 Cuándo escalar a un humano

Desencadenantes de escalada (reglas claras):

Situación	Acción
El cliente solicita explícitamente "dame un gerente"	Escalada inmediata, sin intentar resolver
La política no cubre la solicitud del cliente	Escalada (p.ej., comparación de precios con competencia, política no menciona esto)
El agente no puede hacer progreso	Escalada después de número razonable de intentos

Operación financiera por encima del umbral	Escalada (mejor vía hook que vía prompt)
Múltiples coincidencias al buscar cliente	Solicitar IDs adicionales, no adivinar

Qué NO es desencadenante confiable:

Método no confiable	Por qué no funciona
Análisis de sentimiento	El sentimiento del cliente no se correlaciona con complejidad del caso
Autoevaluación de confianza del modelo (1-10)	El modelo ya está mal calibrado; tiene falsa confianza en decisiones incorrectas
Clasificador automático	Demasiado complejo; requiere datos de entrenamiento que tal vez no existan

9.2 Patrones de escalada

Escalada inmediata:

```
Cliente: "Quiero hablar con un gerente"
Agente: [llama inmediatamente escalate_to_human]
NO: "Puedo ayudarte con tu pregunta, déjame..."
```

Escalada con intento de solución:

```
Cliente: "Mi refrigerador se rompió 2 días después de la compra"
Agente: [verifica pedido, ofrece reemplazo por garantía]
Si cliente no está satisfecho con solución -> escalada
```

Escalada matizada (reconocer → resolver → escalar en reiteración):

```
Cliente: "¡Esto es escandaloso, estoy muy insatisfecho!"
Agente: [reconoce decepción] "Entiendo tu decepción."
       [ofrece solución] "Puedo ofrecer reemplazo o devolución."
Cliente: "No, ¡quiero hablar con alguien!"
Agente: [cliente reitera -> escalada inmediata]
```

Principio clave: primero reconoce emoción del cliente, luego ofrece solución específica, solo con reiteración -- escala. No escales con la primera expresión de descontento (eso no es lo mismo que solicitar gerente).

Escalada por brecha en política:

```
Cliente: "Competidor X vende este producto 30% más barato, dame descuento"
Política: solo describe ajustes de precio para el propio sitio
Agente: [escala -- política no cubre comparación de precios con competencia]
```

9.3 Protocolos estructurados de traspaso (Handoff)

Al escalar, el agente debe pasar al humano un resumen estructurado:

```
{
  "customer_id": "CUST-12345",
  "customer_name": "Juan Pérez",
  "issue_summary": "Solicitud de devolución por producto dañado",
  "order_id": "ORD-67890",
  "root_cause": "Producto llegó dañado, fotos adjuntas",
  "actions_taken": [
    "Cliente verificado vía get_customer",
    "Pedido confirmado vía lookup_order",
    "Se ofreció reemplazo estándar -- cliente insiste en devolución"
  ],
  "refund_amount": "$89.99",
  "recommended_action": "Aprobar devolución completa",
  "escalation_reason": "Cliente solicitó hablar con gerente"
}
```

El operador humano **no tiene acceso al transcrito de conversación** -- solo ve este resumen. Por lo tanto, debe ser completo y autosuficiente.

9.4 Calibración de confianza y control humano

Para sistemas de extracción de datos:

1. **Puntuaciones de confianza a nivel de campo:** el modelo emite puntuación de confianza para cada campo extraído
2. **Calibración:** usar conjuntos de validación etiquetados para ajustar umbrales
3. **Enrutamiento:**
   - Confianza alta + precisión estable -> procesamiento automático
   - Confianza baja o fuente ambigua -> revisión humana

Muestreo estratificado aleatorio:

- Incluso para extracciones de alta confianza, verificar regularmente una muestra
- Precisión agregada del 97% puede ocultar 40% de errores en cierto tipo de documentos
- Analiza precisión por tipo de documento y por campo, no solo en conjunto

---

Capítulo 10: Manejo de errores en sistemas multiagente

10.1 Categorías de errores

Categoría	Ejemplos	¿Reintentar?	Acción del agente
Transient	Timeout, 503, fallo de red	Sí	Reintentar con backoff exponencial
Validation	Formato de entrada inválido, campo obligatorio faltante	No (necesita corregir entrada)	Cambiar solicitud y reintentar
Business	Violación de política, límite excedido	No	Explicar al usuario, ofrecer alternativa
Permission	Sin permisos de acceso	No	Escalar

10.2 Antipatrones de manejo de errores

Antipatrón	Problema	Enfoque correcto
Estado genérico "búsqueda no disponible"	Coordinador no puede decidir cómo recuperarse	Devolver tipo de error, solicitud, resultados parciales, alternativas
Supresión silenciosa (resultado vacío = éxito)	Coordinador piensa que no hay coincidencias, cuando realmente hubo fallo	Distinguir explícitamente "sin resultados" de "fallo en búsqueda"
Terminar todo el proceso con un fallo	Pérdida de todos los resultados parciales	Continuar con resultados parciales, anotar brechas
Reintentos infinitos dentro de subagente	Retraso y desperdicio de recursos	Recuperación local (1-2 reintentos), luego propagar al coordinador

10.3 Error estructurado de subagente

```
{
  "status": "partial_failure",
  "failure_type": "timeout",
  "attempted_query": "Impacto de AI en industria musical",
  "partial_results": [
    {"title": "AI Music Generation Report", "url": "...", "relevance": 0.8}
  ],
  "alternative_approaches": [
    "Intentar consulta más específica: 'Herramientas de composición de música con
AI'",
    "Usar fuente de datos alternativa"
  ],
  "coverage_impact": "Sin cobertura: impacto de AI en producción musical"
}
```

El coordinador obtiene toda la información para decidir:

- ¿Reintentar con solicitud modificada?
- ¿Usar resultados parciales?
- ¿Invitar otro subagente?
- ¿Continuar sin esta sección y anotar brecha?

10.4 Anotaciones de cobertura en síntesis final

```
## Reporte: Impacto de AI en industrias creativas

### Artes visuales (COBERTURA COMPLETA)
[resultados de investigación]

### Música (COBERTURA PARCIAL -- timeout en búsqueda)
[resultados parciales]
⚠️ Nota: la cobertura de esta sección está limitada debido a timeout del agente de
búsqueda.

### Literatura (COBERTURA COMPLETA)
[resultados de investigación]
```

Capítulo 11: Gestión de contexto en sistemas de producción

11.1 Extracción de hechos en bloque separado

En lugar de confiar en el historial de conversación (que se degrada al resumir), extrae hechos clave en un bloque estructurado:

```
=== CASE FACTS (actualizado con cada nuevo hecho) ===
Customer ID: CUST-12345
Order ID: ORD-67890
Order Date: 2025-01-15
Order Amount: $89.99
Issue: Producto dañado en entrega
Customer Request: Devolución completa
Status: Pendiente de aprobación de gerente
===
```

Este bloque se incluye en **cada** prompt, independientemente de la sumarización del historial.

11.2 Recorte de resultados de herramientas

Si `lookup_order` devuelve 40+ campos pero solo necesitas 5 para la tarea actual:

```python
# Hook PostToolUse: mantener solo campos relevantes
@hook("PostToolUse", tool="lookup_order")
def trim_order_fields(result):
    return {
        "order_id": result["order_id"],
        "status": result["status"],
        "total": result["total"],
        "items": result["items"],
        "return_eligible": result["return_eligible"]
    }
```

Esto ahorra ventana de contexto y reduce ruido.

11.3 Entrada consciente de posición

Coloca información crítica considerando el efecto "lost-in-the-middle":

```
[HALLAZGOS CLAVE -- al inicio]
Se descubrieron 3 vulnerabilidades críticas...

[RESULTADOS DETALLADOS -- en el medio]
=== Archivo auth.ts ===
...
=== Archivo database.ts ===
...

[INSTRUCCIONES DE ACCIÓN -- al final]
Prioridad: arregla vulnerabilidades en auth.ts antes de merge.
```

11.4 Archivos Scratchpad

Durante investigación larga de base de código, el agente puede escribir hallazgos clave en archivo scratchpad:

```
# investigation-scratchpad.md
## Hallazgos clave
- Clase PaymentProcessor en src/payments/processor.ts hereda de BaseProcessor
- Método refund() llamado desde 3 lugares: OrderController, AdminPanel, CronJob
- API externo PaymentGateway tiene límite de tasa de 100 req/min
- Migración #47 agregó campo refund_reason (NOT NULL) -- 2024-12-01
```

Al degradarse el contexto (o en sesión nueva), el agente referencia scratchpad en lugar de reinvestigar.

11.5 Delegación a subagentes para proteger contexto

```
Agente principal: "Investiga dependencias del módulo de pagos"
  -> Subagente (Explore): lee 15 archivos, rastrea importes
  -> Devuelve: "Módulo de pagos depende de AuthService, OrderModel, y API externo
PaymentGateway"

Agente principal: guarda 1 línea en lugar de 15 archivos en contexto
```

Capa de contexto separada: En sistemas multiagente, cada subagente trabaja con presupuesto de contexto limitado -- recibe solo información necesaria para su tarea. El coordinador actúa como capa de contexto separada: agrega resultados de subagentes, mantiene estado global, distribuye contexto. Esto previene "fuga de contexto" donde un agente consume la ventana con información irrelevante para otros.

Presupuestos de contexto limitados para subagentes:

- Pasa al subagente contexto mínimo: tarea específica + datos necesarios
- Instruye al subagente devolver solo resultado estructurado, no datos crudos
- Usa `allowedTools` para limitar conjunto de herramientas del subagente -- menos herramientas = menos distracciones y menos gasto de contexto

11.6 Preservación estructurada de estado (para recuperación de crash)

Cada agente exporta su estado a ubicación conocida:

```
// agent-state/web-search-agent.json
{
  "status": "completed",
  "queries_executed": ["AI music 2024", "AI music composition"],
  "results_count": 12,
  "key_findings": [...],
  "coverage": ["music composition", "music production"],
  "gaps": ["music distribution", "music licensing"]
}
```

El coordinador carga el manifiesto al reanudar:

```
// agent-state/manifest.json
{
  "web-search": "completed",
  "doc-analysis": "in_progress",
  "synthesis": "not_started"
}
```

Capítulo 12: Preservación de origen de información (Provenance)

12.1 Problema de pérdida de atribución

Al sumarizar resultados de múltiples fuentes, se pierde conexión "afirmación -> fuente":

```
❌ Malo: "El mercado de AI en música se valúa en $3.2B."
   (¿De dónde viene este número? ¿De qué fuente? ¿Para qué año?)

✅ Bien:
{
  "claim": "El mercado de AI en música se valúa en $3.2B.",
  "source_url": "https://example.com/report",
  "source_name": "Reporte Global de Música con AI 2024",
  "publication_date": "2024-06-15",
  "confidence": 0.9
}
```

12.2 Manejo de datos en conflicto

Cuando dos fuentes dan números diferentes:

```
{
  "claim": "Porcentaje de música generada por AI en plataformas de streaming",
  "values": [
    {
      "value": "12%",
      "source": "Reporte Anual Spotify 2024",
      "date": "2024-03",
      "methodology": "Clasificación automática"
    },
    {
      "value": "8%",
      "source": "Encuesta Asociación Industria Musical",
      "date": "2024-07",
      "methodology": "Encuesta de 500 sellos"
    }
  ],
  "conflict_detected": true,
  "possible_explanation": "Diferencia en metodología y período de tiempo"
}
```

NO selecciones arbitrariamente un valor. Preserva ambos con atribución y deja que el coordinador decida.

12.3 Inclusión de fechas para interpretación correcta

Sin fechas, diferencias temporales se interpretan como contradicciones:

```
❌ "Fuente A dice 10%, fuente B dice 15%. Contradicción."
✅ "Fuente A (2023) dice 10%, fuente B (2024) dice 15%. Probable crecimiento del 5% por año."
```

12.4 Renderizado por tipo de contenido

No conviertas todo a formato único:

- **Datos financieros** -> tablas
- **Noticias y análisis** -> prosa
- **Hallazgos técnicos** -> listas estructuradas
- **Series de tiempo** -> orden cronológico

Capítulo 13: Herramientas integradas de Claude Code

13.1 Guía de referencia para selección de herramientas

Tarea	Herramienta	Ejemplo
Encontrar archivos por nombre/patrón	**Glob**	`**/*.test.tsx`, `src/components/**/*.ts`
Encontrar contenido en archivos	**Grep**	Nombre de función, mensaje de error, import
Leer archivo completo	**Read**	Cargar archivo para análisis
Escribir archivo nuevo	**Write**	Crear archivo nuevo desde cero
Edición puntual de archivo existente	**Edit**	Reemplazar fragmento específico por texto único
Ejecutar comando shell	**Bash**	git, npm, ejecutar pruebas, construir

13.2 Estrategia de exploración incremental

No leas todos los archivos a la vez. Construye la comprensión incrementalmente:

```
1. Grep: encontrar puntos de entrada (definición de función, export)
2. Read: leer archivos encontrados
3. Grep: encontrar usos (import, llamadas)
4. Read: leer archivos consumidores
5. Repetir hasta construir comprensión completa
```

13.3 Fallback: Read + Write en lugar de Edit

Cuando Edit falla por texto no único:

1. **Read** -- cargar contenido completo del archivo
2. Modificar contenido programáticamente
3. **Write** -- escribir versión actualizada

PARTE II: RESUMEN POR DOMINIOS DEL EXAMEN

Dominio 1: Arquitectura de agentes y orquestación (27%)

1.1 Diseño de ciclos de agentes para ejecución autónoma de tareas

Conocimientos clave:

- **Ciclo de vida del ciclo de agente**: enviar solicitud a Claude, verificar `stop_reason` (`"tool_use"` vs `"end_turn"`), ejecutar herramientas, devolver resultados para próxima iteración
- Los resultados de herramientas se agregan al historial de conversación para que el modelo razone sobre la próxima acción
- **Toma de decisiones del modelo** (Claude decide qué herramienta llamar) vs árboles de decisión predefinidos

Habilidades clave:

- Implementar control de flujo: ciclo continúa en `stop_reason = "tool_use"` y finaliza en `"end_turn"`
- Agregar resultados de herramientas al contexto entre iteraciones
- **Antipatrones a evitar**: analizar texto del asistente para determinar finalización, establecer límites arbitrarios de iteraciones como mecanismo de parada principal

1.2 Orquestación de sistemas multiagente (coordinador-subagente)

Conocimientos clave:

- **Arquitectura hub-and-spoke**: coordinador gestiona toda comunicación interagente, manejo de errores y enrutamiento de información
- Los subagentes trabajan con **contexto aislado** -- NO heredan automáticamente historial de conversación del coordinador
- Rol del coordinador: descomposición de tareas, delegación, agregación de resultados, selección dinámica de subagentes
- Riesgo de descomposición excesivamente granular de tareas por coordinador

Habilidades clave:

- Dividir cobertura de investigación entre subagentes para minimizar duplicación
- Implementar ciclos iterativos de refinamiento (coordinador evalúa síntesis, redirige tareas)
- Enrutar toda comunicación a través del coordinador para observabilidad

1.3 Configuración de invocación de subagentes, transferencia de contexto y generación

Conocimientos clave:

- Herramienta `Task` para generar subagentes; `allowedTools` debe incluir `"Task"` para coordinador
- El contexto del subagente **debe pasarse explícitamente** en el prompt -- los subagentes no heredan contexto padre
- Configuración `AgentDefinition`: descripciones, prompts del sistema, restricciones de herramientas
- Gestión de sesiones mediante `fork_session` para explorar enfoques alternativos

Habilidades clave:

- Incluir resultados completos de agentes anteriores en prompt del subagente
- Usar formatos estructurados para separar datos y metadatos al transferir contexto
- Generar subagentes paralelos mediante múltiples llamadas `Task` en una respuesta del coordinador
- Diseñar prompts del coordinador con objetivos y criterios de calidad, no instrucciones paso a paso

1.4 Implementación de flujos de trabajo multisaltos con patrones de aplicación y transferencia

Conocimientos clave:

- Diferencia entre **aplicación programática** (hooks, precondiciones) y **orientación de prompts** para ordenar flujos de trabajo
- Cuando se necesita garantía determinística (verificar identidad antes de operaciones financieras) -- los prompts solos no son suficientes
- Protocolos estructurados de transferencia en escalada (ID cliente, razón, acción recomendada)

Habilidades clave:

- Precondiciones programáticas que bloquean llamadas descendentes hasta finalizar pasos anteriores (bloquear `process_refund` hasta que `get_customer` devuelva ID)
- Descomponer solicitudes multiaspecto de clientes en elementos separados
- Formar resúmenes estructurados al escalar a humano

1.5 Hooks de Agent SDK para interceptar llamadas de herramientas y normalizar datos

Conocimientos clave:

- Patrones de hooks (p.ej., `PostToolUse`) para interceptar resultados de herramientas antes del procesamiento del modelo
- Hooks para interceptar llamadas salientes y aplicar reglas de conformidad (bloquear reembolsos por encima del umbral)

- Los hooks proporcionan **garantías determinísticas** vs instrucciones de prompts dan **conformidad probabilística**

Habilidades clave:

- Hooks `PostToolUse` para normalizar formatos de datos (timestamps Unix, ISO 8601, códigos de estado numéricos)
- Hooks de intercepción para bloquear acciones que violen política con redirección a escalada
- Elegir hooks sobre prompts cuando reglas de negocio requieren conformidad garantizada

1.6 Estrategias de descomposición de tareas para flujos de trabajo complejos

Conocimientos clave:

- **Pipelines fijos** (prompt chaining) vs **descomposición adaptativa dinámica** basada en resultados intermedios
- Prompt chaining: pasos secuenciales (analizar cada archivo por separado, luego pase de integración)
- Planes de investigación adaptativos que generan subtareas basadas en hallazgos

Habilidades clave:

- Prompt chaining para revisiones multisalto predecibles, descomposición dinámica para investigaciones abiertas
- Dividir revisiones grandes de código en análisis por-archivo + pase de integración entre archivos separado
- Descomponer tareas abiertas: primero mapeo de estructura, luego plan priorizado

1.7 Gestión de estado de sesión, reanudación y forking

Conocimientos clave:

- `--resume <session-name>` para continuar sesiones nombradas
- `fork_session` para crear ramas independientes de investigación desde base común
- Importancia de informar al agente sobre cambios de archivo al reanudar sesiones
- Nueva sesión con resumen estructurado más confiable que reanudar con resultados desactualizados

Habilidades clave:

- Usar `--resume` para continuar sesiones de investigación nombradas
- `fork_session` para comparar enfoques en paralelo
- Elegir entre reanudar (contexto aún actual) y nueva sesión (resultados desactualizados)

Dominio 2: Diseño de herramientas e integración MCP (18%)

2.1 Diseño de interfaces de herramientas con descripciones claras

Conocimientos clave:

- Las descripciones de herramientas son el **mecanismo principal** por el cual LLM elige herramienta; descripciones mínimas llevan a selección poco confiable
- Importancia de incluir formatos de entrada, ejemplos de consultas, casos límite y límites de aplicabilidad
- Descripciones ambiguas o superpuestas causan enrutamiento erróneo
- Las formulaciones en el prompt del sistema pueden crear asociaciones no intencionadas con herramientas

Habilidades clave:

- Escribir descripciones que distingan claramente cada herramienta de alternativas similares
- Renombrar herramientas para eliminar superposición funcional (p.ej., `analyze_content` -> `extract_web_results`)
- Dividir herramientas generales en especializadas con contratos específicos de entrada/salida

2.2 Implementación de respuestas de error estructuradas para herramientas MCP

Conocimientos clave:

- Bandera `isError` en MCP para comunicar errores al agente
- Distinción entre **errores transitorios** (timeouts), **errores de validación** (entrada inválida), **errores de negocio** (violación de política) y **errores de acceso**
- Respuestas de error genéricas ("Operation failed") no permiten al agente tomar decisiones correctas de recuperación
- Distinción entre errores repetibles y no repetibles

Habilidades clave:

- Devolver metadatos estructurados: `errorCategory` (transient/validation/permission), `isRetryable`, descripción
- `retriable: false` para violaciones de reglas de negocio con explicaciones comprensibles para usuario
- Recuperación local en subagentes para fallos transitorios, propagación solo de errores no resolubles
- Distinguir fallos de acceso (necesita reintento) de resultados válidos vacíos (sin coincidencias)

2.3 Distribución de herramientas entre agentes y configuración de tool_choice

Conocimientos clave:

- Demasiadas herramientas para un agente (18 en lugar de 4-5) **reduce confiabilidad** de selección
- Agentes con herramientas fuera de su especialización tienden a usarlas incorrectamente
- Acceso de herramientas limitado: solo herramientas para su rol + limitadas entre roles para necesidades comunes
- `tool_choice`: `"auto"`, `"any"`, selección forzada (`{"type": "tool", "name": "..."}`)

Habilidades clave:

- Limitar conjunto de herramientas de cada subagente a las relevantes para su rol
- Reemplazar herramientas generales por alternativas limitadas (p.ej., `fetch_url` -> `load_document`)
- `tool_choice: "any"` para garantizar invocación de herramienta en lugar de respuesta de texto
- Selección forzada de herramienta específica para asegurar orden de ejecución

2.4 Integración de servidores MCP en Claude Code y flujos de trabajo de agentes

Conocimientos clave:

- **Alcance de servidores MCP**: proyecto (`.mcp.json`) para equipo vs usuario (`~/.claude.json`) para experimentos
- Sustitución de variables de entorno en `.mcp.json` (p.ej., `${GITHUB_TOKEN}`) para gestión de secretos
- Las herramientas de todos los servidores MCP conectados se descubren al conectar y están disponibles simultáneamente
- Recursos MCP como catálogos de contenido (resúmenes de tareas, esquemas de BD) para reducir llamadas exploratorias

Habilidades clave:

- Configurar servidores MCP comunes en `.mcp.json` del proyecto con variables de entorno para tokens
- Servidores personales/experimentales en `~/.claude.json`
- Preferir servidores MCP comunitarios existentes sobre propios para integraciones estándar

2.5 Selección y aplicación de herramientas integradas (Read, Write, Edit, Bash, Grep, Glob)

Conocimientos clave:

- **Grep** -- búsqueda de contenido de archivos (nombres de funciones, mensajes de error, imports)

- **Glob** -- búsqueda de archivos por patrones de nombres y extensiones
- **Read/Write** -- operaciones de archivo completo; **Edit** -- cambios puntuales con coincidencia de texto único
- Cuando Edit falla por coincidencia no única -- Read + Write como alternativa

Habilidades clave:

- Grep para búsqueda por contenido en base de código, Glob para encontrar archivos por patrones
- Construir comprensión incrementalmente: Grep para buscar puntos de entrada, luego Read para rastrear flujos
- Rastrear uso de funciones a través de módulos envolventes

Dominio 3: Configuración y flujos de trabajo de Claude Code (20%)

3.1 Configuración de CLAUDE.md con jerarquía, alcance y organización modular

Conocimientos clave:

- **Jerarquía de CLAUDE.md**: usuario (`~/.claude/CLAUDE.md`), proyecto (`.claude/CLAUDE.md` o raíz `CLAUDE.md`), directorio (archivos `CLAUDE.md` en subdirectorios)
- Las configuraciones de usuario se aplican solo a este usuario y no se transmiten vía VCS
- Sintaxis `@path` para referencias a archivos externos (p.ej., `@./standards/coding-style.md`), modularidad de CLAUDE.md
- Directorio `.claude/rules/` para archivos de reglas temáticas en lugar de CLAUDE.md monolítico

Habilidades clave:

- Diagnosticar problemas de jerarquía (nuevo miembro no obtiene instrucciones -- están en nivel usuario en lugar de proyecto)
- `@path` (p.ej., `@./standards/testing.md`) para inclusión selectiva de estándares en CLAUDE.md de cada paquete
- Dividir CLAUDE.md grandes en archivos en `.claude/rules/` (testing.md, api-conventions.md, deployment.md)

3.2 Creación y configuración de comandos slash personalizados y habilidades

Conocimientos clave:

- **Comandos de proyecto** en `.claude/commands/` (vía VCS) vs **personales** en `~/.claude/commands/`

- Habilidades en `.claude/skills/` con archivos `SKILL.md` con frontmatter: `context: fork`, `allowed-tools`, `argument-hint`
- `context: fork` ejecuta habilidad en subagente aislado, sin contaminar sesión principal
- Variantes personales de habilidades en `~/.claude/skills/` con nombres diferentes

Habilidades clave:

- Comandos slash de proyecto en `.claude/commands/` para accesibilidad de todo el equipo
- `context: fork` para aislar habilidades con salida verbosa
- `allowed-tools` para restringir acceso a herramientas al ejecutar habilidad
- `argument-hint` para solicitar parámetros al desarrollador

3.3 Aplicación de reglas específicas de ruta para carga condicional de convenciones

Conocimientos clave:

- Archivos `.claude/rules/` con campo `paths` de frontmatter YAML para activación condicional de reglas por patrones glob
- Reglas de ruta específicas se cargan **solo al editar archivos que coinciden**, ahorrando contexto y tokens
- Ventaja de patrones glob sobre CLAUDE.md a nivel de directorio para convenciones que abarcan múltiples catálogos (p.ej., archivos de prueba)

Habilidades clave:

- Crear archivos `.claude/rules/` con `paths: ["terraform/**/*"]` para carga solo al trabajar con archivos coincidentes
- Patrones glob (`**/*.test.tsx`) para aplicar convenciones por tipo de archivo independientemente de ubicación
- Elegir reglas específicas de ruta sobre CLAUDE.md de directorio cuando convenciones abarcan archivos en toda la base de código

---

Dominio 4: Ingeniería de prompts y salida estructurada (20%)

4.1 Diseño de prompts con criterios explícitos para precisión mejorada

Conocimientos clave:

- Los criterios explícitos importan más que instrucciones vagas (p.ej., "marcar comentarios solo con discrepancia respecto código" vs "verificar precisión de comentarios")

- Las instrucciones genéricas tipo "sé más conservador" funcionan peor que criterios categoriales específicos
- Impacto de falsos positivos en confianza de desarrolladores: alta tasa de falsos positivos en algunas categorías socava confianza en categorías precisas

Habilidades clave:

- Criterios específicos de revisión: qué problemas reportar (bugs, seguridad) vs saltar (estilo menor)
- Deshabilitar temporalmente categorías con alta tasa de falsos positivos
- Definir criterios explícitos de severidad con ejemplos de código para cada nivel

Este es un resumen completo de la guía de certificación Claude Certified Architect, traducida del ruso al español.

El documento incluye teoría fundamental, configuración práctica, estrategias avanzadas y dominios de examen.

Capítulo 3: Claude Agent SDK -- construcción de sistemas de agentes

> Documentación: Agent SDK | Hooks | Subagentes | Sesiones

3.1 ¿Qué es el ciclo agente (Agentic Loop)?

El ciclo agente es el patrón fundamental para la ejecución autónoma de tareas. El modelo no simplemente responde a una pregunta, sino que ejecuta una secuencia de acciones:

```
1. Enviar solicitud a Claude con herramientas
2. Obtener respuesta
3. Verificar stop_reason:
   - "tool_use" -> ejecutar herramienta, agregar resultado al historial, ir al paso 1
   - "end_turn" -> tarea completada, mostrar resultado al usuario
4. Repetir hasta completar
```

Este es un enfoque impulsado por el modelo (model-driven): Claude decide por sí mismo qué herramienta llamar a continuación, basándose en el contexto y los resultados de las acciones anteriores. Esto lo diferencia de los árboles de decisión predefinidos, donde la secuencia de acciones está codificada.

Antipatrones (qué evitar):

- Parsear el texto del asistente para determinar la finalización ("Tarea completada" en el texto de respuesta)
- Establecer un límite arbitrario de iteraciones (max_iterations=5) como mecanismo principal de parada
- Verificar el contenido textual de la respuesta como indicador de finalización

Enfoque correcto: la única señal confiable de finalización es `stop_reason == "end_turn"`.

3.2 Configuración de AgentDefinition

`AgentDefinition` es el objeto de configuración de un agente en Claude Agent SDK:

```
agent = AgentDefinition(
    name="customer_support",
    description="Maneja solicitudes de clientes sobre devoluciones y problemas de
pedidos",
    system_prompt="Eres un agente de soporte al cliente...",
    allowed_tools=["get_customer", "lookup_order", "process_refund",
"escalate_to_human"],
    # Para un coordinador:
    # allowed_tools=["Task", "get_customer", ...]
)
```

Parámetros clave:

- `name` / `description` -- identificación y descripción del agente
- `system_prompt` -- prompt del sistema con instrucciones
- `allowed_tools` -- lista de herramientas permitidas (principio de privilegios mínimos)

3.3 Hub-and-spoke: coordinador y subagentes

La arquitectura multiagente se construye según el principio "hub-and-spoke" (topología de estrella):

```
        Coordinador
       /     |      \
Subagente1  Subagente2  Subagente3
(búsqueda)  (análisis)   (síntesis)
```

El coordinador es responsable de:

- Descomposición de la tarea en subtareas
- Decisión sobre qué subagentes se necesitan (selección dinámica, no siempre todo el pipeline)
- Delegación de tareas a subagentes
- Agregación y validación de resultados
- Manejo de errores y reintentos
- Comunicación de resultados al usuario

Principio crítico: los subagentes tienen contexto aislado.

- Los subagentes **NO heredan** el historial de conversación del coordinador automáticamente
- Todo el contexto debe ser **explícitamente transmitido** en el prompt del subagente
- Los subagentes no comparten memoria entre llamadas
- Toda la comunicación ocurre a través del coordinador (para observabilidad y control de errores)

3.4 Herramienta Task para engendrar subagentes

Los subagentes se generan a través de la herramienta `Task`:

```
# allowed_tools del coordinador debe incluir "Task"
coordinator_agent = AgentDefinition(
    allowed_tools=["Task", "get_customer"]
)
```

La transmisión explícita de contexto es obligatoria:

```
# Incorrecto: el subagente no conoce el contexto
Task: "Analiza el documento"

# Correcto: contexto completo en el prompt
Task: "Analiza el siguiente documento.
Documento: [texto completo del documento]
Resultados de búsqueda anterior: [resultados de búsqueda web]
Requisitos de formato de salida: [esquema]"
```

Generación paralela: el coordinador puede llamar múltiples `Task` en una respuesta -- los subagentes se lanzarán en paralelo:

```
# Una respuesta del coordinador contiene:
Task 1: "Buscar artículos sobre tema X"
Task 2: "Analizar documento Y"
Task 3: "Buscar artículos sobre tema Z"
# Los tres se ejecutarán simultáneamente
```

3.5 Hooks en Agent SDK

Los hooks son un mecanismo para interceptar y transformar en determinados puntos del ciclo de vida del agente.

PostToolUse -- intercepta el resultado de la herramienta antes de pasarlo al modelo:

```
# Ejemplo: normalización de formatos de fecha de diferentes herramientas MCP
@hook("PostToolUse")
def normalize_dates(tool_result):
    # Convertir Unix timestamp -> ISO 8601
    # Convertir "Mar 5, 2025" -> "2025-03-05"
    return normalized_result
```

Hook de interceptación de llamadas salientes -- bloquea acciones que violan políticas:

```python
# Ejemplo: bloquear devoluciones mayores a $500
@hook("PreToolUse")
def enforce_refund_limit(tool_call):
    if tool_call.name == "process_refund" and tool_call.args.amount > 500:
        return redirect_to_escalation(tool_call)
```

Distinción clave: hooks vs instrucciones en prompts

Característica	Hooks	Instrucciones en prompts
Garantía	**Determinística** (100%)	**Probabilística** (>90%, pero no 100%)
Cuándo usar	Reglas comerciales críticas, operaciones financieras, cumplimiento	Preferencias generales, recomendaciones, formateo
Ejemplo	Bloquear devoluciones >$500	"Intenta resolver el problema antes de escalar"

Regla: cuando un error tiene consecuencias financieras, legales o de seguridad -- usa hooks, no prompts.

Capítulo 4: Model Context Protocol (MCP)

> Documentación: MCP | Herramientas | Recursos | Servidores

4.1 ¿Qué es MCP?

Model Context Protocol (MCP) es un protocolo abierto para conectar sistemas externos a Claude. MCP define tres tipos principales de recursos:

1. **Herramientas (Tools)** -- funciones que el agente puede llamar para ejecutar acciones (operaciones CRUD, solicitudes a APIs, ejecución de comandos)
2. **Recursos (Resources)** -- datos a los que el agente puede acceder para obtener contexto (documentación, esquemas de BD, catálogos de contenido)
3. **Prompts (Prompts)** -- plantillas de prompts preestablecidas para tareas típicas

4.2 Servidores MCP

Un servidor MCP es un proceso que implementa el protocolo MCP y proporciona herramientas/recursos. Al conectar a un servidor MCP:

- Todas las herramientas se descubren automáticamente
- Las herramientas de todos los servidores conectados están disponibles simultáneamente
- Las descripciones de las herramientas determinan cómo las usará el modelo

4.3 Configuración de servidores MCP

Configuración del proyecto (`.mcp.json`) -- para uso en equipo:

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${GITHUB_TOKEN}"
      }
    },
    "jira": {
      "command": "npx",
      "args": ["-y", "mcp-server-jira"],
      "env": {
        "JIRA_TOKEN": "${JIRA_TOKEN}"
      }
    }
  }
}
```

Aspectos clave:

- El archivo `.mcp.json` se almacena en la raíz del proyecto y se gestiona a través de VCS
- Se utilizan variables de entorno (`${GITHUB_TOKEN}`) para secretos -- los tokens en sí no se commitean
- Está disponible para todos los miembros del proyecto

Configuración del usuario (`~/.claude.json`) -- para servidores personales/experimentales:

- Se almacena en el directorio home del usuario
- No se transmite a través de VCS
- Apto para experimentos personales y pruebas

Selección de servidores:

- Para integraciones estándar (Jira, GitHub, Slack) -- prefiere servidores MCP comunitarios existentes
- Servidores propios -- solo para flujos de trabajo únicos específicos del equipo

4.4 Bandera isError en MCP

Cuando hay un error en una herramienta MCP, se usa la bandera `isError: true` en la respuesta. Esto señala al agente que la llamada falló.

Error estructurado (correcto):

```
{
  "isError": true,
  "content": {
    "errorCategory": "transient",
    "isRetryable": true,
    "message": "El servicio no está disponible temporalmente. Timeout al acceder a
la API de pedidos.",
    "attempted_query": "order_id=12345",
    "partial_results": null
  }
}
```

Error genérico (antipatrón):

```
{
  "isError": true,
  "content": "Operation failed"
}
```

Un error genérico no proporciona información al agente para tomar decisiones -- ¿reintentar? ¿cambiar la solicitud? ¿escalar?

4.5 Recursos MCP (Resources)

Los recursos son datos que el agente puede solicitar para obtener contexto sin ejecutar acciones:

- Catálogos de contenido (lista de todas las tareas en el proyecto, navegación jerárquica)
- Esquemas de bases de datos (comprensión de la estructura de datos)
- Documentación (referencia de API, guías internas)
- Resúmenes de problemas/tareas

Ventaja de los recursos: el agente no necesita hacer llamadas de herramientas exploratorias para entender los datos disponibles. Un recurso proporciona un "mapa" de inmediato.

Capítulo 5: Claude Code -- configuración y flujos de trabajo

> Documentación: Claude Code | Memoria / CLAUDE.md | Skills | MCP | Hooks | Subagentes | GitHub Actions | Sin interfaz

5.1 Jerarquía de CLAUDE.md

CLAUDE.md es un archivo (o archivos) con instrucciones para Claude Code. Existe una jerarquía con tres niveles:

```
1. Usuario: ~/.claude/CLAUDE.md
   - Se aplica solo a este usuario
   - NO se transmite a través de VCS
   - Preferencias personales, estilo de trabajo

2. Proyecto: .claude/CLAUDE.md o CLAUDE.md en la raíz
   - Se aplica a todos los miembros del proyecto
   - Gestionado a través de VCS
   - Estándares de codificación, pruebas, decisiones arquitectónicas

3. Directorio: CLAUDE.md en subdirectorios
   - Se aplica al trabajar con archivos en este directorio
   - Convenciones específicas para esta parte de la base de código
```

Error típico: un nuevo miembro del equipo no recibe las instrucciones del proyecto porque se colocaron en `~/.claude/CLAUDE.md` (nivel de usuario) en lugar de `.claude/CLAUDE.md` (nivel de proyecto).

5.2 Sintaxis @path (importación de archivos)

CLAUDE.md puede referenciar archivos externos usando sintaxis `@path`, haciendo la configuración modular:

```
# CLAUDE.md del proyecto

Los estándares de codificación se describen en @./standards/coding-style.md
Los requisitos de pruebas están en @./standards/testing-requirements.md
Descripción del proyecto en @README.md y dependencias en @package.json
```

Reglas de sintaxis `@path`:

- Usa `@` directamente antes de la ruta del archivo (sin espacio)
- Se soportan rutas relativas y absolutas
- Las rutas relativas se resuelven relativas al archivo que las contiene
- Profundidad máxima de importación anidada -- 5 niveles

Esto evita duplicación y permite que cada paquete incluya solo los estándares relevantes.

5.3 Directorio .claude/rules/

`.claude/rules/` es una alternativa a un CLAUDE.md monolítico para organizar reglas por temas:

```
.claude/rules/
  testing.md          -- convenciones de pruebas
  api-conventions.md  -- convenciones de API
  deployment.md       -- reglas de despliegue
  react-patterns.md   -- patrones React
```

Característica clave: Frontmatter YAML con campo `paths` para carga condicional:

```
---
paths: ["src/api/**/*"]
---

Para archivos API usa async/await con manejo explícito de errores.
Cada endpoint debe devolver un envoltorio de respuesta estándar.
```

```
---
paths: ["**/*.test.tsx", "**/*.test.ts"]
---

Las pruebas deben usar bloques describe/it.
Usa factories de datos en lugar de hardcode.
No mockees la base de datos -- usa una BD de prueba.
```

Cómo funciona:

- La regla se carga **solo** cuando Claude Code edita un archivo que coincide con el patrón `paths`
- Esto ahorra contexto y tokens -- las reglas irrelevantes no se cargan
- Los patrones glob permiten aplicar convenciones por tipo de archivo **independientemente de la ubicación** -- ideal para pruebas dispersas en toda la base de código

Cuándo usar `.claude/rules/` con paths vs CLAUDE.md a nivel de directorio:

- `.claude/rules/` con paths -- cuando las convenciones se aplican a archivos dispersos en muchos directorios (pruebas, migraciones)
- CLAUDE.md a nivel de directorio -- cuando las convenciones están vinculadas a un directorio específico y no se necesitan fuera de él

5.4 Comandos slash personalizados y Skills

> **Nota:** en la versión actual de Claude Code, los comandos personalizados (`.claude/commands/`) se han fusionado con las habilidades (`.claude/skills/`). Ambos formatos crean comandos invocables mediante `/nombre`. La guía del examen hace referencia a `.claude/commands/` -- este formato aún se soporta.

Los comandos slash son plantillas de prompts reutilizables invocables mediante `/nombre`:

Formato mediante `.claude/commands/` (legado, soportado):

```
.claude/commands/
  review.md        -- /review -- revisión de código estándar
  test-gen.md      -- /test-gen -- generación de pruebas
```

Formato mediante `.claude/skills/` (actual):

```
.claude/skills/
  review/SKILL.md  -- /review -- con configuración en frontmatter
  test-gen/SKILL.md
```

Comandos del proyecto (`.claude/commands/` o `.claude/skills/`):

- Se almacenan en VCS, disponibles para todos al clonar el repositorio
- Proporcionan uniformidad en los flujos de trabajo del equipo

Comandos personalizados (`~/.claude/commands/` o `~/.claude/skills/`):

- Comandos personales, no transmitidos a través de VCS
- Para flujos de trabajo individuales

5.5 Habilidades (Skills) -- .claude/skills/

Las habilidades son comandos extendidos con configuración mediante frontmatter de SKILL.md:

```yaml
---
context: fork
allowed-tools: ["Read", "Grep", "Glob"]
argument-hint: "Ruta del directorio a analizar"
---

Analiza la estructura de código en el directorio especificado.
Genera un reporte sobre dependencias y patrones arquitectónicos.
```

Parámetros de frontmatter:

Parámetro	Descripción
`context: fork`	Ejecuta la habilidad en un subagente aislado. La salida detallada no contamina la sesión principal
`allowed-tools`	Restringe las herramientas disponibles (seguridad -- la habilidad no puede eliminar archivos si no está permitido)
`argument-hint`	Pista que solicita un parámetro al invocar sin argumentos

Cuándo usar habilidad vs CLAUDE.md:

- **Habilidad** -- invocación bajo demanda para una tarea específica (revisión, análisis, generación)
- **CLAUDE.md** -- estándares y convenciones universales siempre cargados

Habilidades personales (`~/.claude/skills/`):

- Crea variantes personales con nombres diferentes para no afectar a colegas

5.6 Modo de planificación (Plan Mode) vs ejecución directa

Modo de planificación:

- El modelo **solo** explora y planifica, sin realizar cambios
- Utiliza Read, Grep, Glob para explorar la base de código
- Resultado -- un plan de implementación, aprobado por el usuario
- Exploración segura sin efectos secundarios

Cuándo usar modo de planificación:

- Cambios a gran escala (docenas de archivos)
- Múltiples enfoques válidos (microservicios: ¿cómo dividir los límites?)
- Decisiones arquitectónicas (¿qué framework? ¿qué estructura?)
- Base de código desconocida (necesita comprensión antes del cambio)
- Migraciones de librerías que afectan 45+ archivos

Cuándo usar ejecución directa:

- Correcciones de un solo archivo con stack trace claro
- Agregar una sola verificación de validación
- Cambios bien entendidos e inequívocos

Enfoque combinado:

1. Modo de planificación para exploración y diseño
2. Aprobación del plan por el usuario
3. Ejecución directa para implementar el plan aprobado

Subagente explore -- subagente especializado para explorar la base de código:

- Aísla la salida detallada del contexto principal
- Devuelve solo un resumen
- Previene el agotamiento de la ventana de contexto en tareas multifase

5.7 Comando /compact

`/compact` es un comando integrado para comprimir el contexto:

- Sumariza el historial anterior, liberando espacio en la ventana de contexto
- Se utiliza en sesiones largas de investigación cuando el contexto se llena con salida detallada de herramientas
- Riesgo: se pierden valores numéricos exactos, fechas y detalles concretos durante la sumarización

5.8 Comando /memory

`/memory` es un comando integrado para gestionar la memoria entre sesiones:

- Abre el archivo `CLAUDE.md` para edición, permitiendo guardar notas, preferencias y contexto
- La información se preserva entre sesiones y se carga automáticamente al iniciar
- Útil para almacenar: convenciones del proyecto, preferencias del usuario, comandos frecuentes, contexto del trabajo actual
- Alternativa a repetir las mismas instrucciones en cada sesión

5.9 Claude Code CLI para CI/CD

Bandera `-p` (o `--print`):

```
claude -p "Analiza este pull request por problemas de seguridad"
```

- Modo no interactivo: procesa el prompt, imprime en stdout, finaliza
- Sin espera de entrada del usuario
- **La única forma correcta** de ejecutarse en pipelines CI/CD

Salida estructurada para CI:

```
claude -p "Revisa este PR" --output-format json --json-schema
'{"type":"object",...}'
```

- `--output-format json` -- salida en formato JSON
- `--json-schema` -- validación de salida según esquema
- El resultado se puede parsear para postear automáticamente comentarios inline en PRs

Aislamiento de contexto de sesión: La misma sesión de Claude que **generó** el código es menos efectiva para su **revisión** (el modelo retiene el contexto de razonamiento y es menos propenso a cuestionar sus decisiones). Usa una **instancia independiente** para la revisión.

Prevención de comentarios duplicados: Al revisar nuevamente después de nuevos commits -- incluye los resultados de revisión anteriores en el contexto e instruye a Claude que reporte solo problemas **nuevos o no corregidos**.

5.10 fork_session y gestión de sesiones

`--resume <session-name>` -- reanuda una sesión nombrada:

```
claude --resume investigation-auth-bug
```

- Continúa la conversación anterior con contexto guardado
- Útil para investigaciones largas en múltiples sesiones
- Riesgo: si los archivos han cambiado desde la sesión anterior, los resultados de las herramientas pueden estar obsoletos

`fork_session` -- crear una rama independiente desde una base común:

```
Exploración de base de código
          |
    fork_session
    /           \
Enfoque A:      Enfoque B:
Redux          Context API
```

- Ambas ramas heredan el contexto hasta el punto de bifurcación
- Luego se desarrollan independientemente
- Útil para comparar enfoques o probar estrategias

Cuándo iniciar una nueva sesión en lugar de reanudar:

- Los resultados de las herramientas están obsoletos (los archivos cambiaron)
- Pasó mucho tiempo y el contexto se degradó
- Es mejor comenzar con "Aquí hay un resumen de lo que descubrimos: ..." que reanudar con datos antiguos

---

Capítulo 6: Ingeniería de prompts -- técnicas avanzadas

> Documentación: Ingeniería de prompts | Anthropic Cookbook

6.1 Prompting Few-shot

Few-shot prompting incluye 2-4 ejemplos de entrada/salida en el prompt para demostrar el comportamiento esperado.

Por qué few-shot es más efectivo que descripciones textuales:

- Una descripción textual "sé más preciso" se interpreta de diferentes formas
- Un ejemplo muestra inequívocamente el formato esperado y la lógica de decisión
- El modelo generaliza el patrón a casos nuevos (no simplemente repite ejemplos)

Tipos de ejemplos few-shot y su aplicación:

1. **Ejemplos para escenarios ambiguos:**

```
Solicitud: "Mi pedido está roto"
Acción: Llamar get_customer -> lookup_order -> verificar estado.
Justificación: "roto" puede significar producto dañado. Necesita verificar detalles del
pedido.

Solicitud: "Dame un gerente"
Acción: Llamar inmediatamente escalate_to_human.
Justificación: El cliente solicita explícitamente a una persona. No intentes resolver
solo.
```

2. **Ejemplos para formato de salida:**

```
Hallazgo de ejemplo:
{
  "location": "src/auth/login.ts:42",
  "issue": "SQL injection en parámetro username",
  "severity": "critical",
  "suggested_fix": "Usar consulta parametrizada"
}
```

3. **Ejemplos para distinguir código válido de problemático:**

```
// Válido (no marcar):
const items = data.filter(x => x.active);

// Problema (marcar):
const items = data.filter(x => x.active == true); // Usa comparación estricta ===
```

4. **Ejemplos para extracción de diferentes formatos de documento:**

```
Documento con citas inline:
"Como se muestra en el estudio (Smith, 2023), el nivel es del 42%."
-> {"value": "42%", "source": "Smith, 2023", "type": "inline_citation"}

Documento con bibliografía:
"El nivel es del 42%. [1]"
-> {"value": "42%", "source": "reference_1", "type": "bibliography"}
```

5. **Ejemplos para medidas informales (informal measurements):**

```
Texto: "aproximadamente dos puñados de arroz"
-> {"amount": "~100g", "original_text": "dos puñados", "precision": "approximate"}

Texto: "una pizca de sal"
-> {"amount": "~1g", "original_text": "pizca", "precision": "approximate"}
```

Few-shot es especialmente efectivo para extraer medidas informales y no estándar, donde las reglas textuales no pueden cubrir toda la variedad de expresiones.

Reglas de normalización de formato en prompts: Al usar esquemas JSON estrictos para salida estructurada, agrega reglas de normalización de formato al prompt:

```
Normalización:
- Fechas: siempre ISO 8601 (YYYY-MM-DD), "ayer" -> calcular fecha absoluta
- Moneda: valor numérico + código de moneda, "cinco dólares" -> {"amount": 5,
"currency": "USD"}
- Porcentajes: decimal, "mitad" -> 0.5
```

Esto previene errores semánticos cuando el esquema JSON es sintácticamente válido pero los valores son inconsistentes.

6.2 Criterios explícitos vs instrucciones vagas

Incorrecto (vago):

```
Verifica comentarios en el código por precisión.
Sé conservador, reporta solo hallazgos de alta confianza.
```

Correcto (criterios explícitos):

```
Marca un comentario como problemático SOLO si:
1. El comentario describe comportamiento que CONTRADICE el comportamiento real del
código
2. El comentario referencia una función o variable que no existe
3. Un comentario TODO/FIXME para un bug que ya fue corregido en el código

NO marques:
- Comentarios que simplemente están anticuados estilísticamente
- Comentarios con inexactitudes menores en la formulación
- Ausencia de comentarios (esa es otra categoría)
```

Definición de criterios de severidad con ejemplos:

```
CRÍTICO: Fallo de runtime para usuarios
  Ejemplo: NullPointerException al procesar pago

ALTO: Vulnerabilidad de seguridad
  Ejemplo: SQL injection, XSS, falta de verificación de permisos

MEDIO: Error lógico sin efecto inmediato
  Ejemplo: Ordenamiento incorrecto, error off-by-one

BAJO: Calidad del código
  Ejemplo: Duplicación, algoritmo no óptimo para datos pequeños
```

6.3 Prompt Chaining (cadenas de prompts)

Prompt chaining es dividir una tarea compleja en pasos secuenciales enfocados:

```
Paso 1: Análisis del archivo auth.ts (solo problemas locales)
       -> Resultado: lista de problemas en auth.ts

Paso 2: Análisis del archivo database.ts (solo problemas locales)
       -> Resultado: lista de problemas en database.ts

Paso 3: Pase de integración (dependencias entre archivos)
       -> Resultado: problemas en interfaces de módulos
```

Por qué se necesita:

- Evita **dilución de atención (attention dilution)** -- cuando el modelo recibe demasiados archivos a la vez, puede perder bugs en unos mientras da comentarios superficiales en otros
- Proporciona **calidad consistente** del análisis para cada archivo
- Permite **análisis separado** de interacciones entre archivos

Cuándo usar prompt chaining vs descomposición dinámica:

- **Prompt chaining** -- para tareas predecibles y repetitivas (revisión de código, migración de archivos)
- **Descomposición dinámica** -- para investigaciones abiertas donde las subtareas se vuelven claras solo durante el proceso

6.4 Patrón "entrevista"

Antes de implementar una solución, Claude hace preguntas aclaratorias al desarrollador:

```
Claude: "Antes de implementar el caché para la API, algunas preguntas:
1. ¿Qué estrategia de invalidación de caché es preferible -- TTL o event-based?
2. ¿Es aceptable tener datos obsoletos (stale) si el caché no está disponible?
3. ¿Necesita cachear a nivel de usuarios individuales o globalmente?
4. ¿Cuál es el volumen de datos esperado para cachear?"
```

Cuándo es útil:

- Dominio desconocido (fintech, medicina, sistemas legales)
- Tarea con implicaciones poco obvias (estrategias de caché, modos de fallo)
- Múltiples enfoques válidos, cuya selección depende del contexto

6.5 Validación y retry-with-feedback

Cuando los datos extraídos no pasan la validación:

```
Paso 1: Extracción de datos del documento
Paso 2: Validación (Pydantic, JSON Schema, reglas comerciales)
Paso 3: Si hay error -- reintento con contexto:
  - Documento original
  - Extracción anterior (errónea)
  - Error específico: "Campo 'total' = 150, pero suma de line_items = 145. Verifica
valores."
```

Cuándo el reintento será efectivo:

- Errores de formato (fecha en formato incorrecto)
- Errores de estructura (campo en lugar incorrecto)
- Discrepancias aritméticas (el modelo puede reverificar)

Cuándo el reintento NO ayudará:

- Información ausente en la fuente (no está en documento -- el reintento no la agregará)
- Contexto externo (datos en otro documento no transmitido al modelo)

Pydantic como herramienta de validación: Pydantic es una librería Python para validación de datos basada en esquemas. En el contexto del examen, lo importante es:

- **Validación de estructura:** Pydantic verifica tipos de campos, obligatoriedad, valores permitidos (enum) a nivel de código después de obtener JSON de Claude
- **Validación semántica:** Validadores personalizados verifican lógica comercial (suma de posiciones = total, fecha inicio < fecha fin)
- **Ciclos validación/reintento:** Al error de validación Pydantic -- formamos mensaje de error y enviamos a Claude solicitud de reintento con contexto de error
- **Generación de esquemas JSON:** Los modelos Pydantic pueden generar JSON Schema automáticamente para el parámetro `tool_use`, proporcionando una única fuente de verdad para el esquema

6.6 Auto-corrección (Self-correction)

Patrón para detectar contradicciones internas:

```
{
  "stated_total": "$150.00",
  "calculated_total": "$145.00",
  "conflict_detected": true,
  "line_items": [
    {"name": "Widget A", "price": 75.00},
    {"name": "Widget B", "price": 70.00}
  ]
}
```

El modelo extrae **tanto** el valor declarado **como** el calculado -- si difieren, la bandera `conflict_detected` permite manejar la divergencia.

Capítulo 7: Message Batches API

> Documentación: Message Batches

7.1 Descripción general

Message Batches API permite enviar paquetes de solicitudes para procesamiento asincrónico:

Característica	Valor
Ahorro	**50%** del costo de llamadas sincrónicas
Ventana de procesamiento	Hasta **24 horas** (sin garantías SLA en latencia)
Llamadas a herramientas multi-turn	**No soportado** (una solicitud = una respuesta)
Correlación	Campo `custom_id` para vincular solicitud y respuesta

7.2 Cuándo usar Batch API vs API sincrónica

Regla: cuando un error tiene consecuencias financieras, legales o de seguridad -- usa hooks, no prompts.

Capítulo 4: Model Context Protocol (MCP)

> Documentación: MCP | Tools | Resources | Servers

4.1 ¿Qué es MCP?

Model Context Protocol (MCP) es un protocolo abierto para conectar sistemas externos a Claude. MCP define tres tipos principales de recursos:

1. **Tools (herramientas)** -- funciones que el agente puede llamar para ejecutar acciones (operaciones CRUD, llamadas API, ejecución de comandos)
2. **Resources (recursos)** -- datos a los que el agente puede acceder para obtener contexto (documentación, esquemas de BD, catálogos de contenido)
3. **Prompts (prompts)** -- plantillas de prompts predefinidas para tareas típicas

---

Dominio 5: Gestión de contexto y confiabilidad (15%)

5.1 Gestión de contexto para preservar información crítica

Conocimientos clave:

- **Riesgos de sumarización progresiva**: condensación de valores numéricos, porcentajes, fechas en resúmenes vagos

- Efecto **"lost-in-the-middle"**: los modelos procesan inicio y final confiablemente, pero pueden perder hallazgos del medio
- Los resultados de herramientas se acumulan en contexto desproporcionalmente a relevancia
- Importancia transmitir historial completo en solicitudes API subsecuentes

Habilidades clave:

- Extraer hechos transaccionales a bloque permanente "case facts"
- Recortar salida verbosa de herramientas a campos relevantes
- Colocar hallazgos clave al inicio de datos agregados
- Requerir a subagentes incluir metadatos (fechas, fuentes)

Examen Práctico

> 76 preguntas en 5 escenarios. El formato y la dificultad coinciden con el examen real.
>
> Como alternativa, puedes practicar estas preguntas en un archivo HTML estilo examen: Examen Práctico (ES)

Escenario: Sistema de investigación multiagente

Pregunta 1 (Escenario: Sistema de investigación multiagente)

Situación: Un agente de análisis de documentos descubre que dos fuentes creíbles contienen estadísticas directamente contradictorias para una métrica clave: un informe gubernamental indica un crecimiento del 40%, mientras que un análisis de la industria indica un 12%. Ambas fuentes parecen creíbles, y la discrepancia podría afectar materialmente las conclusiones de la investigación. ¿Cómo debería el agente de análisis de documentos manejar esta situación de la forma más efectiva?

¿Cuál es el enfoque más efectivo?

- A) Aplicar heurísticas de credibilidad para elegir el número más probablemente correcto, terminar el análisis con ese valor y agregar una nota al pie mencionando la discrepancia.
- B) Incluir ambos números en la salida del análisis sin marcarlos como conflictivos, dejando que el agente de síntesis decida cuál usar según el contexto más amplio.
- C) Detener el análisis y escalar inmediatamente al coordinador, pidiéndole que decida qué fuente es más autoritativa antes de continuar.
- D) Completar el análisis con ambos números, anotar explícitamente el conflicto con atribución de fuente y dejar que el coordinador decida cómo reconciliar los datos antes de pasarlos a síntesis. **[CORRECTA]**

Por qué D: Este enfoque preserva la separación de responsabilidades: el agente de análisis completa su trabajo principal sin bloquearse, conserva ambos valores conflictivos con atribución clara y traslada

correctamente la reconciliación al coordinador, que tiene un contexto más amplio.

Pregunta 2 (Escenario: Sistema de investigación multiagente)

Situación: Los agentes de búsqueda web y de análisis de documentos completaron sus tareas y devolvieron resultados al coordinador. ¿Cuál es el siguiente paso para crear un informe de investigación integrado?

¿Cuál es el siguiente paso más apropiado?

- A) Cada agente envía sus resultados directamente al agente redactor del informe, evitando al coordinador.
- B) El agente de análisis de documentos solicita los resultados de búsqueda web y los fusiona internamente.
- C) El coordinador pasa ambos conjuntos de resultados al agente de síntesis para una integración unificada. **[CORRECTA]**
- D) El coordinador concatena las salidas crudas de ambos agentes y las devuelve como resultado final.

Por qué C: En una arquitectura coordinador–subagente, el coordinador reenvía ambos conjuntos de resultados al agente de síntesis para una integración centralizada, preservando el control y asegurando una fusión de alta calidad.

Pregunta 3 (Escenario: Sistema de investigación multiagente)

Situación: Un subagente de análisis de documentos falla con frecuencia al procesar archivos PDF: algunos tienen secciones corruptas que disparan excepciones de parseo, otros están protegidos con contraseña, y a veces la biblioteca de parseo se cuelga con archivos grandes. Actualmente, cualquier excepción termina inmediatamente el subagente y devuelve un error al coordinador, que debe decidir si reintentar, omitir o fallar toda la tarea. Esto causa una participación excesiva del coordinador en el manejo rutinario de errores. ¿Qué mejora arquitectónica es más efectiva?

¿Qué mejora es más efectiva?

- A) Crear un agente dedicado al manejo de errores que monitoree todos los fallos a través de una cola compartida y decida acciones de recuperación, enviando comandos de reinicio directamente a los subagentes.
- B) Configurar el subagente para que siempre devuelva resultados parciales con estado de éxito, embebiendo detalles del error en metadatos; el coordinador trata todas las respuestas como exitosas.
- C) Hacer que el coordinador valide todos los documentos antes de enviarlos al subagente, rechazando documentos que puedan causar fallos.
- D) Implementar recuperación local en el subagente para fallos transitorios y escalar al coordinador solo los errores que no puede resolver, incluyendo pasos intentados y resultados parciales. **[CORRECTA]**

Por qué D: Maneja los errores en el nivel más bajo capaz de resolverlos. La recuperación local reduce la carga del coordinador mientras escala los problemas verdaderamente irrecuperables con contexto completo y progreso parcial.

Pregunta 4 (Escenario: Sistema de investigación multiagente)

Situación: Después de ejecutar el sistema sobre "el impacto de la IA en las industrias creativas", observas que cada subagente se completa con éxito: el agente de búsqueda web encuentra artículos relevantes, el agente de análisis de documentos los resume correctamente y el agente de síntesis produce un texto coherente. Sin embargo, los informes finales solo cubren artes visuales y omiten por completo música, literatura y cine. En los registros del coordinador, ves que descompuso el tema en tres subtareas: "IA en arte digital", "IA en diseño gráfico" e "IA en fotografía". ¿Cuál es la causa raíz más probable?

¿Cuál es la causa raíz más probable?

- A) Al agente de síntesis le faltan instrucciones para detectar lagunas de cobertura.
- B) El agente de análisis de documentos filtra fuentes no visuales debido a criterios de relevancia demasiado estrictos.
- C) La descomposición de tareas del coordinador es demasiado estrecha, asignando a los subagentes trabajo que no cubre todas las áreas relevantes. **[CORRECTA]**
- D) Las consultas del agente de búsqueda web son insuficientes y deberían ampliarse para cubrir más sectores.

Por qué C: El coordinador descompuso un tema amplio solo en subtareas de artes visuales, omitiendo por completo música, literatura y cine. Como los subagentes ejecutaron sus asignaciones correctamente, la descomposición estrecha es la causa raíz evidente.

Pregunta 5 (Escenario: Sistema de investigación multiagente)

Situación: El subagente de búsqueda web devuelve resultados solo para 3 de 5 categorías de fuentes solicitadas (sitios de competencia e informes de la industria tienen éxito, pero archivos de noticias y feeds sociales agotan el tiempo). El subagente de análisis de documentos procesa con éxito todos los documentos provistos. El subagente de síntesis debe producir un resumen a partir de entradas previas de calidad mixta. ¿Qué estrategia de propagación de errores es más efectiva?

¿Qué estrategia de propagación de errores es más efectiva?

- A) Continuar la síntesis usando solo las fuentes exitosas y producir una salida sin mencionar qué datos no estaban disponibles.
- B) El subagente de síntesis devuelve un error al coordinador, disparando un reintento completo o el fallo de la tarea por datos incompletos.
- C) El subagente de síntesis pide al coordinador que reintente las fuentes con tiempo de espera agotado con un timeout más largo antes de iniciar la síntesis.
- D) Estructurar la salida de síntesis con anotaciones de cobertura que indiquen qué conclusiones están bien respaldadas y dónde hay vacíos por fuentes no disponibles. **[CORRECTA]**

Por qué D: Las anotaciones de cobertura implementan una degradación elegante con transparencia, preservando el valor del trabajo completado mientras propagan la incertidumbre para permitir decisiones informadas sobre la confianza.

Pregunta 6 (Escenario: Sistema de investigación multiagente)

Situación: El subagente de análisis de documentos encuentra un archivo PDF corrupto que no puede parsear. Al diseñar el manejo de errores del sistema, ¿cuál es la forma más efectiva de manejar este fallo?

¿Cuál es el enfoque más efectivo?

- A) Devolver un error con contexto al agente coordinador, permitiéndole decidir cómo proceder. **[CORRECTA]**
- B) Omitir silenciosamente el documento corrupto y continuar procesando los archivos restantes para no interrumpir el flujo.
- C) Reintentar automáticamente el parseo del documento tres veces con retroceso exponencial antes de reportar un fallo.
- D) Lanzar una excepción que termine todo el flujo de investigación.

Por qué A: Devolver un error con contexto al coordinador es lo más efectivo porque le permite tomar una decisión informada—omitir el archivo, intentar un método de parseo alternativo o notificar al usuario—mientras se mantiene visibilidad sobre el fallo.

Pregunta 7 (Escenario: Sistema de investigación multiagente)

Situación: Los registros de producción muestran un patrón persistente: solicitudes como "analiza el informe trimestral subido" se enrutan al agente de búsqueda web el 45% del tiempo en lugar de al agente de análisis de documentos. Revisando las definiciones de herramientas, encuentras que el agente de búsqueda web tiene una herramienta `analyze_content` descrita como "analiza contenido y extrae información clave", mientras que el agente de análisis de documentos tiene una herramienta `analyze_document` descrita como "analiza documentos y extrae información clave". ¿Cómo deberías corregir el problema de enrutamiento?

¿Cómo deberías corregir el problema de enrutamiento?

- A) Agregar un clasificador de pre-enrutamiento que detecte si el usuario se refiere a archivos subidos o contenido web antes de que el coordinador decida la delegación.
- B) Renombrar la herramienta de búsqueda web a `extract_web_results` y actualizar su descripción a "procesa y devuelve información obtenida de búsqueda web y URLs". **[CORRECTA]**
- C) Agregar ejemplos few-shot al prompt del coordinador mostrando el enrutamiento correcto: "El usuario sube un informe trimestral → agente de análisis de documentos" y "El usuario pregunta sobre una página web → agente de búsqueda web".
- D) Expandir la descripción de la herramienta de análisis de documentos con ejemplos de uso como "Usar para PDFs subidos, documentos de Word y hojas de cálculo", dejando la herramienta de búsqueda web sin cambios.

Por qué B: Renombrar la herramienta de búsqueda web a `extract_web_results` y actualizar su descripción para referenciar explícitamente la búsqueda web y las URLs elimina directamente la causa raíz al eliminar el solapamiento semántico entre los nombres y descripciones de las dos herramientas.

Esto hace inequívoco el propósito de cada herramienta, permitiendo al coordinador distinguir confiablemente análisis de documentos de búsqueda web.

Pregunta 8 (Escenario: Sistema de investigación multiagente)

Situación: Un colega propone que el agente de análisis de documentos envíe sus resultados directamente al agente de síntesis, evitando al coordinador. ¿Cuál es la principal ventaja de mantener al coordinador como hub central de toda la comunicación entre subagentes?

¿Cuál es la principal ventaja de mantener al coordinador como hub central?

- A) El coordinador puede observar todas las interacciones, manejar errores uniformemente y decidir qué información debe recibir cada subagente. **[CORRECTA]**
- B) El coordinador agrupa múltiples solicitudes a los subagentes, reduciendo el total de llamadas a la API y la latencia general.
- C) El enrutamiento a través del coordinador permite lógica de reintento automático que las llamadas directas entre agentes no pueden soportar.
- D) Los subagentes usan memoria aislada, y la comunicación directa requeriría serialización compleja que solo el coordinador puede realizar.

Por qué A: El patrón coordinador proporciona visibilidad centralizada de todas las interacciones, manejo uniforme de errores en todo el sistema y control fino sobre qué información recibe cada subagente—estas son las ventajas principales de una topología de comunicación en estrella.

Pregunta 9 (Escenario: Sistema de investigación multiagente)

Situación: El subagente de búsqueda web agota el tiempo de espera mientras investiga un tema complejo. Necesitas diseñar cómo se devuelve la información sobre este fallo al coordinador. ¿Qué enfoque de propagación de errores permite mejor una recuperación inteligente?

¿Qué enfoque de propagación de errores permite mejor una recuperación inteligente?

- A) Devolver contexto de error estructurado al coordinador, incluyendo el tipo de fallo, la consulta ejecutada, cualquier resultado parcial y posibles enfoques alternativos. **[CORRECTA]**
- B) Capturar el timeout dentro del subagente y devolver un conjunto de resultados vacío marcado como exitoso.
- C) Implementar reintentos automáticos con retroceso exponencial dentro del subagente, devolviendo solo un estado genérico "búsqueda no disponible" después de agotar los reintentos.
- D) Propagar la excepción de timeout directamente al manejador de nivel superior, terminando todo el flujo de investigación.

Por qué A: Devolver contexto de error estructurado—incluyendo tipo de fallo, consulta ejecutada, resultados parciales y enfoques alternativos—da al coordinador todo lo necesario para tomar decisiones inteligentes de recuperación (por ejemplo, reintentar con una consulta modificada o continuar con resultados parciales). Preserva el máximo contexto para una toma de decisiones informada a nivel de coordinación.

Pregunta 10 (Escenario: Sistema de investigación multiagente)

Situación: En tu diseño del sistema, le diste al agente de análisis de documentos acceso a una herramienta de propósito general `fetch_url` para que pudiera descargar documentos por URL. Los registros de producción muestran que este agente ahora descarga frecuentemente páginas de resultados de motores de búsqueda para realizar búsquedas web ad hoc—comportamiento que debería enrutarse a través del agente de búsqueda web—causando resultados inconsistentes. ¿Qué corrección es más efectiva?

¿Qué corrección es más efectiva?

- A) Reemplazar `fetch_url` con una herramienta `load_document` que valide que las URLs apunten a formatos de documento. **[CORRECTA]**
- B) Eliminar `fetch_url` del agente de análisis de documentos y enrutar toda obtención de URL a través del coordinador hacia el agente de búsqueda web.
- C) Implementar un filtro que bloquee llamadas de `fetch_url` a dominios conocidos de motores de búsqueda mientras permite otras URLs.
- D) Agregar instrucciones al prompt del agente de análisis de documentos indicando que `fetch_url` solo debe usarse para descargar URLs de documentos, no para buscar.

Por qué A: Reemplazar una herramienta de propósito general con una herramienta específica para documentos que valida URLs contra formatos de documento corrige la causa raíz limitando la capacidad a nivel de la interfaz. Esto sigue el principio de menor privilegio, haciendo imposible el comportamiento de búsqueda no deseado en lugar de meramente desincentivarlo.

Pregunta 11 (Escenario: Sistema de investigación multiagente)

Situación: Mientras investigas un tema amplio, observas que el agente de búsqueda web y el agente de análisis de documentos investigan los mismos subtemas, llevando a una duplicación sustancial en sus salidas. El uso de tokens casi se duplica sin un aumento proporcional en la amplitud o profundidad de la investigación. ¿Cuál es la forma más efectiva de abordar esto?

¿Cuál es la forma más efectiva de abordar esto?

- A) Permitir que ambos agentes terminen en paralelo y luego que el coordinador deduplique los resultados solapados antes de pasarlos al agente de síntesis.
- B) El coordinador particiona explícitamente el espacio de investigación antes de delegar, asignando a cada agente subtemas o tipos de fuente distintos. **[CORRECTA]**
- C) Implementar un mecanismo de estado compartido donde los agentes registran su área de enfoque actual para que otros agentes puedan evitar dinámicamente la duplicación durante la ejecución.
- D) Cambiar a ejecución secuencial donde el análisis de documentos se ejecuta solo después de que la búsqueda web termina, usando los resultados de búsqueda como contexto para evitar duplicación.

Por qué B: Hacer que el coordinador particione explícitamente el espacio de investigación antes de delegar es lo más efectivo porque aborda la causa raíz—límites de tarea poco claros—antes de que

comience cualquier trabajo. Preserva el paralelismo mientras previene esfuerzo duplicado y tokens desperdiciados.

Pregunta 12 (Escenario: Sistema de investigación multiagente)

Situación: Durante la investigación, el subagente de búsqueda web consulta tres categorías de fuentes con resultados diferentes: las bases de datos académicas devuelven 15 artículos relevantes, los informes de la industria devuelven "0 resultados" y las bases de datos de patentes devuelven "Tiempo de conexión agotado". Al diseñar la propagación de errores al coordinador, ¿qué enfoque permite las mejores decisiones de recuperación?

¿Qué enfoque permite las mejores decisiones de recuperación?

- A) Agregar los resultados en una sola métrica de porcentaje de éxito (por ejemplo, "67% de cobertura de fuentes") con registros detallados disponibles a demanda.
- B) Reportar tanto "timeout" como "0 resultados" como fallos que requieren intervención del coordinador.
- C) Reintentar fallos transitorios internamente y reportar solo errores persistentes.
- D) Distinguir fallos de acceso (timeout) que requieren una decisión de reintento de resultados vacíos válidos ("0 resultados") que representan consultas exitosas. **[CORRECTA]**

Por qué D: Un timeout (fallo de acceso) y "0 resultados" (resultado vacío válido) son resultados semánticamente diferentes que requieren respuestas diferentes. Distinguirlos permite al coordinador reintentar la base de datos de patentes mientras acepta los "0 resultados" de informes de la industria como un hallazgo válido e informativo.

Pregunta 13 (Escenario: Sistema de investigación multiagente)

Situación: El monitoreo de producción muestra calidad de síntesis inconsistente. Cuando los resultados agregados son ~75K tokens, el agente de síntesis cita confiablemente información de los primeros 15K tokens (titulares/fragmentos de búsqueda web) y los últimos 10K tokens (conclusiones del análisis de documentos), pero a menudo se pierde hallazgos críticos en los 50K tokens del medio—incluso cuando responden directamente a la pregunta de investigación. ¿Cómo deberías reestructurar la entrada agregada?

¿Cómo deberías reestructurar la entrada agregada?

- A) Resumir todas las salidas de los subagentes a menos de 20K tokens antes de la agregación para mantener el contenido dentro del rango confiable de procesamiento del modelo.
- B) Transmitir los resultados de los subagentes al agente de síntesis incrementalmente, procesando primero los resultados de búsqueda web por completo, luego agregando los resultados del análisis de documentos.
- C) Colocar un resumen de hallazgos clave al inicio de la entrada agregada y organizar los resultados detallados con encabezados de sección explícitos para una navegación más fácil. **[CORRECTA]**

- D) Implementar rotación que alterne qué resultados de subagente aparecen primero en distintas tareas de investigación para asegurar que ambas fuentes obtengan posicionamiento superior equitativo con el tiempo.

Por qué C: Poner un resumen de hallazgos clave al inicio aprovecha los efectos de primacía para que la información crítica esté en la posición procesada más confiablemente. Agregar encabezados de sección explícitos en todo el documento ayuda al modelo a navegar y atender el contenido del medio, mitigando directamente el fenómeno "perdido en el medio".

Pregunta 14 (Escenario: Sistema de investigación multiagente)

Situación: En pruebas, la salida combinada del agente de búsqueda web (85K tokens incluyendo contenido de página) y del agente de análisis de documentos (70K tokens incluyendo cadenas de pensamiento) totaliza 155K tokens, pero el agente de síntesis funciona mejor con entradas por debajo de 50K tokens. ¿Qué solución es más efectiva?

¿Qué solución es más efectiva?

- A) Modificar los agentes previos para que devuelvan datos estructurados (hechos clave, citas, puntuaciones de relevancia) en lugar de contenido y razonamiento verbosos. **[CORRECTA]**
- B) Agregar un agente intermedio de resumición que condense los hallazgos antes de pasarlos a la síntesis.
- C) Hacer que el agente de síntesis procese los hallazgos en lotes secuenciales, manteniendo el estado entre llamadas.
- D) Almacenar los hallazgos en una base de datos vectorial y dar al agente de síntesis herramientas de búsqueda para consultar durante su trabajo.

Por qué A: Modificar los agentes previos para que devuelvan datos estructurados corrige la causa raíz reduciendo el volumen de tokens en la fuente mientras preserva la información esencial. Evita pasar contenido de página voluminoso y trazas de razonamiento que inflan tokens sin mejorar el paso de síntesis.

Pregunta 15 (Escenario: Sistema de investigación multiagente)

Situación: En pruebas, observas que el agente de síntesis a menudo necesita verificar afirmaciones específicas mientras fusiona resultados. Actualmente, cuando se necesita verificación, el agente de síntesis devuelve el control al coordinador, que llama al agente de búsqueda web y luego reinvoca la síntesis con los resultados. Esto añade 2–3 bucles extra por tarea y aumenta la latencia un 40%. Tu evaluación muestra que el 85% de estas verificaciones son comprobaciones simples de hechos (fechas, nombres, estadísticas) y el 15% requiere investigación más profunda. ¿Qué enfoque reduce más efectivamente la sobrecarga preservando la confiabilidad del sistema?

¿Cuál es el enfoque más efectivo?

- A) Dar al agente de síntesis acceso a todas las herramientas de búsqueda web para que pueda manejar cualquier necesidad de verificación directamente sin bucles del coordinador.

- B) Hacer que el agente de síntesis acumule todas las necesidades de verificación y las devuelva como un lote al coordinador al final, que las envía todas al agente de búsqueda web a la vez.
- C) Hacer que el agente de búsqueda web cachee proactivamente contexto extra alrededor de cada fuente durante la investigación inicial en anticipación a que la síntesis necesite verificación.
- D) Dar al agente de síntesis una herramienta `verify_fact` de alcance limitado para comprobaciones simples, mientras se enrutan las verificaciones complejas a través del coordinador al agente de búsqueda web. **[CORRECTA]**

Por qué D: Una herramienta de verificación de hechos de alcance limitado permite al agente de síntesis manejar el 85% de las comprobaciones simples directamente, eliminando la mayoría de los bucles, mientras se preserva la ruta de delegación del coordinador para el 15% de verificaciones complejas. Esto aplica el menor privilegio mientras reduce significativamente la latencia.

Escenario: Claude Code para Integración Continua

Pregunta 16 (Escenario: Claude Code para Integración Continua)

Situación: Tu pipeline de CI ejecuta el CLI de Claude Code (en modo `--print`) usando CLAUDE.md para proporcionar contexto del proyecto a la revisión de código, y los desarrolladores generalmente encuentran las revisiones sustantivas. Sin embargo, reportan que integrar los hallazgos al flujo es difícil—Claude produce párrafos narrativos que deben copiarse manualmente a los comentarios del PR. El equipo quiere publicar automáticamente cada hallazgo como un comentario inline separado del PR en el lugar relevante del código, lo que requiere datos estructurados con ruta de archivo, número de línea, nivel de severidad y corrección sugerida. ¿Qué enfoque es más efectivo?

¿Cuál es el enfoque más efectivo?

- A) Agregar una sección "Output Format for Review" a CLAUDE.md con ejemplos de hallazgos estructurados para que Claude aprenda el formato esperado del contexto del proyecto.
- B) Usar las flags del CLI `--output-format json` y `--json-schema` para imponer hallazgos estructurados, luego parsear la salida para publicar comentarios inline a través de la API de GitHub. **[CORRECTA]**
- C) Incluir instrucciones de formato explícitas en el prompt de revisión que requieran que cada hallazgo siga una plantilla parseable como `[FILE:ruta] [LINE:n] [SEVERITY:nivel] ...`.
- D) Mantener el formato de revisión narrativo pero agregar un paso de resumición que use Claude para generar un resumen JSON estructurado de los hallazgos.

Por qué B: Usar `--output-format json` con `--json-schema` impone salida estructurada a nivel del CLI, garantizando JSON bien formado con los campos requeridos (ruta de archivo, número de línea, severidad, corrección sugerida) que pueden parsearse y publicarse confiablemente como comentarios inline del PR a través de la API de GitHub. Aprovecha capacidades incorporadas del CLI diseñadas específicamente para salida estructurada.

Pregunta 17 (Escenario: Claude Code para Integración Continua)

Situación: Tu equipo usa Claude Code para generar sugerencias de código, pero notas un patrón: problemas no obvios—optimizaciones de rendimiento que rompen casos límite, limpiezas que cambian inesperadamente el comportamiento—solo se detectan cuando otro miembro del equipo revisa el PR. El razonamiento de Claude durante la generación muestra que consideró estos casos pero concluyó que su enfoque era correcto. ¿Qué enfoque aborda directamente la causa raíz de esta limitación de auto-revisión?

¿Qué enfoque aborda directamente la causa raíz?

- A) Ejecutar una segunda instancia independiente de Claude Code para revisar los cambios sin acceso al razonamiento del generador. **[CORRECTA]**
- B) Habilitar el modo de pensamiento extendido para la etapa de generación para permitir una deliberación más exhaustiva antes de producir sugerencias.
- C) Agregar instrucciones explícitas de auto-revisión al prompt de generación pidiendo a Claude que critique sus propias sugerencias antes de finalizar la salida.
- D) Incluir archivos de prueba completos y documentación en el contexto del prompt para que Claude entienda mejor el comportamiento esperado durante la generación.

Por qué A: Una segunda instancia independiente de Claude Code sin acceso al razonamiento del generador aborda directamente la causa raíz al evitar el sesgo de confirmación. Esta perspectiva de "ojos frescos" refleja la revisión por pares humana, donde otro revisor detecta problemas que el autor racionalizó.

Pregunta 18 (Escenario: Claude Code para Integración Continua)

Situación: Tu componente de revisión de código es iterativo: Claude analiza el archivo modificado, luego puede solicitar archivos relacionados (imports, clases base, pruebas) mediante llamadas a herramientas para entender el contexto antes de proporcionar la retroalimentación final. Tu aplicación define una herramienta que permite a Claude solicitar contenido de archivos; Claude llama la herramienta, obtiene resultados y continúa el análisis. Estás evaluando procesamiento en lote para reducir el costo de la API. ¿Cuál es la principal limitación técnica al considerar procesamiento en lote para este flujo?

¿Cuál es la principal limitación técnica?

- A) El procesamiento en lote no incluye IDs de correlación para mapear las salidas de vuelta a las solicitudes de entrada.
- B) El modelo asíncrono no puede ejecutar herramientas a mitad de solicitud y devolver resultados para que Claude continúe el análisis. **[CORRECTA]**
- C) La Batch API no soporta definiciones de herramientas en los parámetros de solicitud.
- D) La latencia de procesamiento en lote de hasta 24 horas es demasiado lenta para retroalimentación de pull requests, aunque el flujo funcionaría de otro modo.

Por qué B: Un modelo asíncrono "fire-and-forget" de Batch API no tiene mecanismo para interceptar una llamada a herramienta durante una solicitud, ejecutar la herramienta y devolver resultados para que Claude continúe el análisis. Esto es fundamentalmente incompatible con flujos iterativos de llamadas a

herramientas que requieren múltiples rondas de solicitud/respuesta de herramientas dentro de una sola interacción lógica.

Pregunta 19 (Escenario: Claude Code para Integración Continua)

Situación: Tu sistema CI/CD ejecuta tres análisis basados en Claude: (1) verificaciones rápidas de estilo en cada PR que bloquean el merge hasta completarse, (2) auditorías exhaustivas de seguridad semanales de toda la base de código, y (3) generación nocturna de casos de prueba para módulos cambiados recientemente. La Message Batches API ofrece 50% de ahorro pero el procesamiento puede tardar hasta 24 horas. Quieres optimizar el costo de la API manteniendo una experiencia de desarrollador aceptable. ¿Qué combinación empareja correctamente cada tarea con un enfoque de API?

¿Qué combinación es correcta?

- A) Usar la Message Batches API para las tres tareas para maximizar el 50% de ahorro, configurando el pipeline para sondear la finalización del lote.
- B) Usar llamadas síncronas para verificaciones de estilo del PR; usar la Message Batches API para auditorías de seguridad semanales y generación nocturna de pruebas. **[CORRECTA]**
- C) Usar llamadas síncronas para las tres tareas para tiempos de respuesta consistentes, confiando en el caching de prompts para reducir costos en todas las cargas de trabajo.
- D) Usar llamadas síncronas para verificaciones de estilo del PR y generación nocturna de pruebas; usar la Message Batches API solo para auditorías de seguridad semanales.

Por qué B: Las verificaciones de estilo del PR bloquean a los desarrolladores y requieren respuestas inmediatas vía llamadas síncronas, mientras que las auditorías de seguridad semanales y la generación nocturna de pruebas son tareas programadas con plazos flexibles que pueden tolerar hasta una ventana de lote de 24 horas—capturando 50% de ahorro para ambas.

Pregunta 20 (Escenario: Claude Code para Integración Continua)

Situación: Tus revisiones automatizadas encuentran problemas reales, pero los desarrolladores reportan que la retroalimentación no es accionable. Los hallazgos incluyen frases como "lógica de enrutamiento de tickets compleja" o "potencial puntero nulo" sin especificar qué cambiar exactamente. Cuando agregas instrucciones detalladas como "siempre incluir sugerencias de corrección concretas", el modelo aún produce salida inconsistente—a veces detallada, a veces vaga. ¿Qué técnica de prompting produce más confiablemente retroalimentación consistentemente accionable?

¿Qué técnica de prompting es más confiable?

- A) Refinar más las instrucciones con requisitos más explícitos para cada parte del formato de retroalimentación (ubicación, problema, severidad, corrección propuesta).
- B) Expandir la ventana de contexto para incluir más código circundante para que el modelo tenga suficiente información para proponer correcciones concretas.
- C) Implementar un enfoque de dos pasadas donde un prompt identifica problemas y un segundo genera correcciones, permitiendo especialización.

- D) Agregar 3–4 ejemplos few-shot que muestren el formato exacto requerido: problema identificado, ubicación en el código, sugerencia de corrección concreta. **[CORRECTA]**

Por qué D: Los ejemplos few-shot son la técnica más efectiva para lograr formato de salida consistente cuando las instrucciones por sí solas producen resultados variables. Proporcionar 3–4 ejemplos que muestran la estructura exacta deseada (problema, ubicación, corrección concreta) le da al modelo un patrón concreto a seguir, lo cual es más confiable que instrucciones abstractas.

Pregunta 21 (Escenario: Claude Code para Integración Continua)

Situación: Tu pipeline de CI incluye dos modos de revisión de código basados en Claude: un hook de pre-merge-commit que bloquea el merge del PR hasta completarse, y un "análisis profundo" que se ejecuta de noche, sondea la finalización del lote y publica sugerencias detalladas en el PR. Quieres reducir el costo de la API usando la Message Batches API, que ofrece 50% de ahorro pero requiere sondeo y puede tardar hasta 24 horas. ¿Qué modo debería usar procesamiento en lote?

¿Qué modo debería usar procesamiento en lote?

- A) Solo el hook de pre-merge-commit.
- B) Solo el análisis profundo. **[CORRECTA]**
- C) Ambos modos.
- D) Ninguno de los modos.

Por qué B: El análisis profundo es un candidato ideal para procesamiento en lote porque ya se ejecuta de noche, tolera retraso y usa un modelo de sondeo antes de publicar resultados—coincidiendo con la arquitectura asíncrona basada en sondeo de la Message Batches API mientras captura 50% de ahorro.

Pregunta 22 (Escenario: Claude Code para Integración Continua)

Situación: Tu revisión automatizada analiza comentarios y docstrings. El prompt actual instruye a Claude a "verificar que los comentarios sean precisos y estén actualizados". Los hallazgos a menudo señalan patrones aceptables (marcadores TODO, descripciones simples) mientras se pierden comentarios que describen comportamiento que el código ya no implementa. ¿Qué cambio aborda la causa raíz de este análisis inconsistente?

¿Qué cambio aborda la causa raíz?

- A) Incluir datos de `git blame` para que Claude pueda identificar comentarios que preceden cambios de código recientes.
- B) Agregar ejemplos few-shot de comentarios engañosos para ayudar al modelo a reconocer patrones similares en la base de código.
- C) Filtrar patrones de comentarios TODO, FIXME y descriptivos antes del análisis para reducir el ruido.
- D) Especificar criterios explícitos: marcar comentarios solo cuando el comportamiento que afirman contradice el comportamiento real del código. **[CORRECTA]**

Por qué D: Los criterios explícitos—marcar comentarios solo cuando el comportamiento afirmado contradice el comportamiento real del código—abordan directamente la causa raíz reemplazando una instrucción vaga con una definición precisa de qué constituye un problema. Esto reduce los falsos positivos sobre patrones aceptables y los descuidos de comentarios verdaderamente engañosos.

Pregunta 23 (Escenario: Claude Code para Integración Continua)

Situación: Tu sistema automatizado de revisión de código muestra calificaciones de severidad inconsistentes—problemas similares como riesgos de puntero nulo se califican como "críticos" en algunos PRs pero solo "medio" en otros. Las encuestas a desarrolladores muestran creciente desconfianza—muchos comienzan a descartar hallazgos sin leer porque "la mitad están mal". Las categorías con altos falsos positivos erosionan la confianza en categorías precisas. ¿Qué enfoque restaura mejor la confianza del desarrollador mientras mejora el sistema?

¿Qué enfoque restaura mejor la confianza del desarrollador?

- A) Deshabilitar temporalmente categorías con altos falsos positivos (estilo, nomenclatura, documentación) y mantener solo categorías de alta precisión mientras se mejoran los prompts. **[CORRECTA]**
- B) Mantener todas las categorías habilitadas pero mostrar puntuaciones de confianza con cada hallazgo para que los desarrolladores decidan qué investigar.
- C) Mantener todas las categorías habilitadas y agregar ejemplos few-shot para mejorar la precisión de cada categoría durante las próximas semanas.
- D) Aplicar una reducción uniforme de estrictez a todas las categorías para bajar la tasa general de falsos positivos.

Por qué A: Deshabilitar temporalmente las categorías con altos falsos positivos detiene inmediatamente la erosión de confianza al eliminar hallazgos ruidosos que hacen que los desarrolladores descarten todo, mientras se preserva el valor de las categorías de alta precisión como seguridad y corrección. También crea espacio para mejorar los prompts de las categorías problemáticas antes de rehabilitarlas.

Pregunta 24 (Escenario: Claude Code para Integración Continua)

Situación: Tu revisión automatizada genera sugerencias de casos de prueba para cada PR. Revisando un PR que agrega seguimiento de finalización de cursos, Claude sugiere 10 casos de prueba, pero la retroalimentación del desarrollador muestra que 6 duplican escenarios ya cubiertos por la suite de pruebas existente. ¿Qué cambio reduce más efectivamente las sugerencias duplicadas?

¿Qué cambio es más efectivo?

- A) Incluir el archivo de pruebas existente en el contexto para que Claude pueda determinar qué escenarios ya están cubiertos. **[CORRECTA]**
- B) Reducir el número solicitado de sugerencias de 10 a 5, asumiendo que Claude prioriza primero los casos más valiosos.

- C) Agregar instrucciones dirigiendo a Claude a enfocarse exclusivamente en casos límite y condiciones de error en lugar de rutas de éxito.
- D) Implementar post-procesamiento que filtre sugerencias cuyas descripciones coincidan con nombres de pruebas existentes mediante solapamiento de palabras clave.

Por qué A: Incluir el archivo de pruebas existente corrige la causa raíz de la duplicación: Claude solo puede evitar sugerir escenarios ya cubiertos si sabe qué pruebas ya existen. Esto le da a Claude la información necesaria para proponer pruebas genuinamente nuevas y valiosas.

Pregunta 25 (Escenario: Claude Code para Integración Continua)

Situación: Después de que una revisión automatizada inicial identifica 12 hallazgos, un desarrollador hace commits nuevos para abordar problemas. Al volver a ejecutar la revisión se producen 8 hallazgos, pero los desarrolladores reportan que 5 duplican comentarios anteriores sobre código que ya fue corregido en los nuevos commits. ¿Cuál es la forma más efectiva de eliminar esta retroalimentación redundante manteniendo la exhaustividad?

¿Cuál es la forma más efectiva de eliminar la retroalimentación redundante?

- A) Ejecutar la revisión solo cuando se crea el PR y en el estado final pre-merge, omitiendo commits intermedios.
- B) Agregar un filtro de post-procesamiento que elimine hallazgos que coincidan con anteriores por rutas de archivo y descripciones de problemas antes de publicar comentarios.
- C) Restringir el alcance de la revisión a archivos cambiados en el push más reciente, excluyendo archivos de commits anteriores.
- D) Incluir los hallazgos de revisión anteriores en el contexto e instruir a Claude para reportar solo problemas nuevos o aún sin resolver. **[CORRECTA]**

Por qué D: Incluir los hallazgos de revisión previos en el contexto permite a Claude distinguir problemas nuevos de los ya abordados en commits recientes. Esto preserva la exhaustividad de la revisión mientras usa el razonamiento de Claude para evitar retroalimentación redundante sobre código corregido.

Pregunta 26 (Escenario: Claude Code para Integración Continua)

Situación: Tu script de pipeline ejecuta `claude "Analyze this pull request for security issues"`, pero el job se cuelga indefinidamente. Los registros muestran que Claude Code está esperando entrada interactiva. ¿Cuál es el enfoque correcto para ejecutar Claude Code en un pipeline automatizado?

¿Cuál es el enfoque correcto?

- A) Agregar una flag `--batch`: `claude --batch "Analyze this pull request for security issues"`.
- B) Agregar la flag `-p`: `claude -p "Analyze this pull request for security issues"`. **[CORRECTA]**

- C) Redirigir stdin desde `/dev/null`: `claude "Analyze this pull request for security issues" < /dev/null`.
- D) Establecer la variable de entorno `CLAUDE_HEADLESS=true` antes de ejecutar el comando.

Por qué B: La flag `-p` (o `--print`) es la forma documentada de ejecutar Claude Code de forma no interactiva. Procesa el prompt, imprime el resultado a stdout y sale sin esperar entrada del usuario—ideal para pipelines de CI/CD.

Pregunta 27 (Escenario: Claude Code para Integración Continua)

Situación: Un pull request cambia 14 archivos en un módulo de seguimiento de inventario. Una revisión de una sola pasada que analiza todos los archivos juntos produce resultados inconsistentes: retroalimentación detallada en algunos archivos pero comentarios superficiales en otros, errores obvios omitidos y retroalimentación contradictoria (un patrón se marca en un archivo pero código idéntico se aprueba en otro archivo del mismo PR). ¿Cómo deberías reestructurar la revisión?

¿Cómo deberías reestructurar la revisión?

- A) Ejecutar tres pasadas independientes de revisión completa del PR y marcar solo problemas que aparezcan en al menos dos de las tres ejecuciones.
- B) Dividir en pasadas enfocadas: revisar cada archivo individualmente para problemas locales, luego ejecutar una pasada separada orientada a la integración para examinar flujos de datos entre archivos. **[CORRECTA]**
- C) Requerir que los desarrolladores dividan PRs grandes en envíos más pequeños de 3–4 archivos antes de ejecutar la revisión automatizada.
- D) Cambiar a un modelo más grande con una ventana de contexto mayor para que pueda prestar atención suficiente a los 14 archivos en una sola pasada.

Por qué B: Las pasadas enfocadas por archivo abordan la causa raíz—dilución de atención—asegurando profundidad consistente y detección confiable de problemas locales. Una pasada separada orientada a la integración cubre luego preocupaciones entre archivos como dependencias e interacciones de flujo de datos.

Pregunta 28 (Escenario: Claude Code para Integración Continua)

Situación: Tu revisión automatizada de código promedia 15 hallazgos por pull request, y los desarrolladores reportan una tasa de falsos positivos del 40%. El cuello de botella es el tiempo de investigación: los desarrolladores deben hacer clic en cada hallazgo para leer la justificación de Claude antes de decidir si corregir o descartar. Tu CLAUDE.md ya contiene reglas exhaustivas para patrones aceptables, y los stakeholders rechazaron cualquier enfoque que filtre hallazgos antes de que los vean los desarrolladores. ¿Qué cambio aborda mejor el tiempo de investigación?

¿Qué cambio aborda mejor el tiempo de investigación?

- A) Requerir que Claude incluya su justificación y estimación de confianza directamente en cada hallazgo. **[CORRECTA]**

- B) Agregar un post-procesador que analice patrones de hallazgos y suprima automáticamente aquellos que coincidan con firmas históricas de falsos positivos.
- C) Categorizar los hallazgos como "problemas bloqueantes" vs "sugerencias", con diferentes requisitos de revisión por nivel.
- D) Configurar Claude para mostrar solo hallazgos de alta confianza, filtrando marcadores inciertos antes de que los vean los desarrolladores.

Por qué A: Incluir la justificación y la confianza directamente en cada hallazgo reduce el tiempo de investigación al permitir que los desarrolladores triien rápidamente sin abrir cada hallazgo. Satisface la restricción de "no filtrar" porque todos los hallazgos permanecen visibles mientras se acelera la toma de decisiones del desarrollador.

Pregunta 29 (Escenario: Claude Code para Integración Continua)

Situación: El análisis de tu revisión automatizada de código muestra grandes diferencias en las tasas de falsos positivos por categoría de hallazgo: hallazgos de seguridad/corrección tienen 8% de falsos positivos, hallazgos de rendimiento 18%, hallazgos de estilo/nomenclatura 52% y hallazgos de documentación 48%. Las encuestas a desarrolladores muestran creciente desconfianza—muchos comienzan a descartar hallazgos sin leer porque "la mitad están mal". Las categorías con altos falsos positivos erosionan la confianza en categorías precisas. ¿Qué enfoque restaura mejor la confianza del desarrollador mientras mejora el sistema?

¿Qué enfoque restaura mejor la confianza del desarrollador?

- A) Deshabilitar temporalmente categorías con altos falsos positivos (estilo, nomenclatura, documentación) y mantener solo categorías de alta precisión mientras se mejoran los prompts. **[CORRECTA]**
- B) Mantener todas las categorías habilitadas pero mostrar puntuaciones de confianza con cada hallazgo para que los desarrolladores decidan qué investigar.
- C) Mantener todas las categorías habilitadas y agregar ejemplos few-shot para mejorar la precisión de cada categoría durante las próximas semanas.
- D) Aplicar una reducción uniforme de estrictez a todas las categorías para bajar la tasa general de falsos positivos.

Por qué A: Deshabilitar temporalmente las categorías con altos falsos positivos detiene inmediatamente la erosión de confianza al eliminar hallazgos ruidosos que hacen que los desarrolladores descarten todo, mientras se preserva el valor de las categorías de alta precisión como seguridad y corrección. También crea espacio para mejorar los prompts de las categorías problemáticas antes de rehabilitarlas.

Pregunta 30 (Escenario: Claude Code para Integración Continua)

Situación: Tu equipo quiere reducir los costos de API para análisis automatizado. Actualmente, las llamadas síncronas a Claude soportan dos flujos: (1) una verificación pre-merge bloqueante que debe completarse antes de que los desarrolladores puedan hacer merge, y (2) un informe de deuda técnica

generado durante la noche para revisión a la mañana siguiente. Tu gerente propone mover ambos a la Message Batches API para ahorrar 50%. ¿Cómo deberías evaluar esta propuesta?

¿Cómo deberías evaluar esta propuesta?

- A) Mover ambos a procesamiento en lote con respaldo a llamadas síncronas si los lotes tardan demasiado.
- B) Mover ambos flujos a procesamiento en lote con sondeo de estado para verificar la finalización.
- C) Usar procesamiento en lote solo para los informes de deuda técnica; mantener las llamadas síncronas para las verificaciones pre-merge. **[CORRECTA]**
- D) Mantener llamadas síncronas para ambos flujos para evitar problemas con el ordenamiento de resultados de lotes.

Por qué C: El procesamiento de la Message Batches API puede tardar hasta 24 horas sin SLA de latencia, lo cual es aceptable para informes nocturnos de deuda técnica pero inaceptable para verificaciones pre-merge bloqueantes donde los desarrolladores esperan. Esto empareja cada flujo con la API correcta según los requisitos de latencia.

Escenario: Generación de código con Claude Code

Pregunta 31 (Escenario: Generación de código con Claude Code)

Situación: Le pediste a Claude Code que implementara una función que transforme respuestas de API a un formato interno normalizado. Después de dos iteraciones, la estructura de salida aún no coincide con las expectativas—algunos campos están anidados de forma diferente y las marcas de tiempo están formateadas incorrectamente. Describiste los requisitos en prosa, pero Claude los interpreta de forma diferente cada vez.

¿Qué enfoque es más efectivo para la siguiente iteración?

- A) Escribir un esquema JSON que describa la estructura de salida esperada y validar la salida de Claude contra él después de cada iteración.
- B) Proporcionar 2–3 ejemplos concretos de entrada-salida que muestren la transformación esperada para respuestas de API representativas. **[CORRECTA]**
- C) Reescribir los requisitos con mayor precisión técnica, especificando mapeos exactos de campos, reglas de anidamiento y cadenas de formato de marca de tiempo.
- D) Pedir a Claude que explique su comprensión actual de los requisitos para identificar dónde divergen las interpretaciones.

Por qué B: Los ejemplos concretos de entrada-salida eliminan la ambigüedad inherente a las descripciones en prosa al mostrar a Claude los resultados exactos de transformación esperados. Esto aborda directamente la causa raíz—mala interpretación de requisitos textuales—proporcionando patrones inequívocos para anidamiento de campos y formato de marcas de tiempo.

Pregunta 32 (Escenario: Generación de código con Claude Code)

Situación: Necesitas agregar Slack como un nuevo canal de notificación. La base de código existente tiene patrones claros y establecidos para canales de email, SMS y push. Sin embargo, la API de Slack ofrece enfoques de integración fundamentalmente diferentes—webhooks entrantes (simple, unidireccional), bot tokens (soportan confirmación de entrega y control programático) o Slack Apps (eventos bidireccionales, requiere aprobación del workspace). Tu tarea dice "agregar soporte para Slack" sin especificar el método de integración ni requerir características avanzadas como seguimiento de entrega.

¿Cómo deberías abordar esta tarea?

- A) Comenzar en modo de ejecución directa usando webhooks entrantes para coincidir con el patrón existente de notificación unidireccional.
- B) Cambiar a modo de planificación para explorar opciones de integración e implicaciones arquitectónicas, luego presentar una recomendación antes de implementar. **[CORRECTA]**
- C) Comenzar en modo de ejecución directa esbozando una clase de canal Slack usando los patrones existentes, posponiendo la decisión del método de integración.
- D) Comenzar en modo de ejecución directa usando un enfoque de bot token para asegurar que sea posible la confirmación de entrega.

Por qué B: La integración con Slack tiene múltiples enfoques válidos con implicaciones arquitectónicas significativamente diferentes, y los requisitos son ambiguos. El modo de planificación te permite evaluar trade-offs entre webhooks, bot tokens y Slack Apps y alinearte sobre un enfoque antes de la implementación.

Pregunta 33 (Escenario: Generación de código con Claude Code)

Situación: Tu archivo CLAUDE.md ha crecido a más de 400 líneas conteniendo estándares de codificación, convenciones de pruebas, una checklist detallada de revisión de PRs, instrucciones de despliegue y procedimientos de migración de base de datos. Quieres que Claude siga siempre los estándares de codificación y convenciones de pruebas, pero que aplique la guía de revisión de PR, despliegue y migración solo cuando esté haciendo esas tareas.

¿Qué enfoque de reestructuración es más efectivo?

- A) Mover toda la guía a archivos Skills separados organizados por tipo de flujo, dejando solo una breve descripción del proyecto en CLAUDE.md.
- B) Mantener todo en CLAUDE.md pero usar sintaxis `@import` para organizar en archivos mantenidos por separado por categoría.
- C) Dividir CLAUDE.md en archivos bajo `.claude/rules/` con patrones glob ligados a rutas para que cada regla cargue solo para los tipos de archivo relevantes.
- D) Mantener los estándares universales en CLAUDE.md y crear Skills para guía específica de flujo (revisión de PR, despliegue, migraciones) con palabras clave de activación. **[CORRECTA]**

Por qué D: El contenido de CLAUDE.md se carga en cada sesión, asegurando que los estándares de codificación y las convenciones de pruebas siempre apliquen, mientras que las Skills se invocan bajo demanda cuando Claude detecta palabras clave de activación—ideal para guía específica de flujo como revisión de PR, despliegue y migraciones.

Pregunta 34 (Escenario: Generación de código con Claude Code)

Situación: Estás encargado de reestructurar la aplicación monolítica de tu equipo en microservicios. Esto impacta cambios en docenas de archivos y requiere decisiones sobre límites de servicio y dependencias entre módulos.

¿Qué enfoque deberías elegir?

- A) Cambiar a modo de planificación para explorar la base de código, entender dependencias y diseñar el enfoque de implementación antes de hacer cambios. **[CORRECTA]**
- B) Comenzar en modo de ejecución directa y cambiar a planificación solo después de encontrar complejidad inesperada durante la implementación.
- C) Comenzar en modo de ejecución directa y hacer cambios incrementales, dejando que la implementación revele los límites naturales de servicio.
- D) Usar ejecución directa con instrucciones detalladas previas que especifiquen la estructura de cada servicio.

Por qué A: El modo de planificación es la estrategia correcta para reestructuración arquitectónica compleja como dividir un monolito: permite exploración segura y decisiones informadas sobre límites antes de comprometerse a cambios potencialmente costosos en muchos archivos.

Pregunta 35 (Escenario: Generación de código con Claude Code)

Situación: Tu equipo creó un skill `/analyze-codebase` que realiza análisis profundo de código—escaneo de dependencias, conteos de cobertura de pruebas y métricas de calidad de código. Después de ejecutar el comando, los miembros del equipo reportan que Claude se vuelve menos receptivo en la sesión y pierde el contexto de la tarea original.

¿Cómo lo corriges más efectivamente manteniendo capacidades completas de análisis?

- A) Agregar `context: fork` en el frontmatter del skill para ejecutar el análisis en un contexto de subagente aislado. **[CORRECTA]**
- B) Agregar `model: haiku` en el frontmatter para usar un modelo más rápido y económico para el análisis.
- C) Dividir el skill en tres skills más pequeños, cada uno produciendo menos salida.
- D) Agregar instrucciones al skill para comprimir todos los resultados en un resumen corto antes de mostrarlos.

Por qué A: `context: fork` ejecuta el análisis en un contexto de subagente aislado para que la salida grande no contamine la ventana de contexto de la sesión principal y Claude no pierda el rastro de la tarea original. Preserva la capacidad completa de análisis manteniendo la sesión principal receptiva.

Pregunta 36 (Escenario: Generación de código con Claude Code)

Situación: Tu equipo usa un skill `/commit` en `.claude/skills/commit/SKILL.md`. Un desarrollador quiere personalizarlo para su flujo personal (formato de mensaje de commit diferente, verificaciones extra) sin afectar a sus compañeros.

¿Qué recomiendas?

- A) Crear una versión personal bajo `~/.claude/skills/` con un nombre diferente, por ejemplo `/my-commit`.
- B) Agregar lógica condicional basada en nombre de usuario en el frontmatter del skill del proyecto.
- C) Crear una versión personal en `~/.claude/skills/commit/SKILL.md` con el mismo nombre. **[CORRECTA]**
- D) Establecer `override: true` en el frontmatter del skill personal para priorizarlo sobre la versión del proyecto.

Por qué C: Los skills personales tienen precedencia sobre los skills del proyecto con el mismo nombre. Un skill personal en `~/.claude/skills/commit/SKILL.md` sobreescribirá el skill del proyecto, permitiendo al desarrollador personalizar su flujo mientras mantiene el nombre familiar `/commit` para uso personal. Este enfoque es mejor que la opción A porque preserva el nombre del comando original, mejorando el flujo del desarrollador sin afectar a los compañeros.

Pregunta 37 (Escenario: Generación de código con Claude Code)

Situación: Tu equipo ha usado Claude Code durante meses. Recientemente, tres desarrolladores reportan que Claude sigue la guía "siempre incluir manejo de errores exhaustivo", pero un cuarto desarrollador que acaba de unirse dice que Claude no la sigue. Los cuatro trabajan en el mismo repo y tienen código actualizado.

¿Cuál es la causa más probable y la corrección?

- A) La guía vive en los archivos `~/.claude/CLAUDE.md` a nivel de usuario de los desarrolladores originales, no en el `.claude/CLAUDE.md` del proyecto. Mover la instrucción al archivo a nivel de proyecto para que todos los miembros del equipo la reciban. **[CORRECTA]**
- B) El `~/.claude/CLAUDE.md` del nuevo desarrollador contiene instrucciones conflictivas que sobrescriben la configuración del proyecto; debería eliminar la sección conflictiva.
- C) Claude Code aprende preferencias por usuario con el tiempo; el nuevo desarrollador debe repetir el requisito hasta que Claude lo "recuerde".

- D) Claude Code cachea CLAUDE.md después de la primera lectura; los desarrolladores originales usan versiones cacheadas. Todos deberían limpiar el caché de Claude Code.

Por qué A: Si la guía se agregó solo a las configuraciones a nivel de usuario de los desarrolladores originales y no al `.claude/CLAUDE.md` a nivel de proyecto, los nuevos miembros del equipo no la recibirán. Moverla a la configuración a nivel de proyecto asegura que todos los miembros del equipo actuales y futuros reciban automáticamente la guía.

Pregunta 38 (Escenario: Generación de código con Claude Code)

Situación: Encuentras que incluir 2–3 ejemplos completos de implementación de endpoints como contexto mejora significativamente la consistencia al generar nuevos endpoints de API. Sin embargo, este contexto solo es útil al crear nuevos endpoints—no al depurar, revisar código u otros trabajos en el directorio API.

¿Qué enfoque de configuración es más efectivo?

- A) Agregar ejemplos de endpoints y documentación de patrones al CLAUDE.md del proyecto para que estén siempre disponibles.
- B) Referenciar manualmente los ejemplos de endpoints en cada solicitud de generación copiando el código al prompt.
- C) Configurar reglas específicas por ruta en `.claude/rules/api/` que incluyan ejemplos de endpoints y se activen al trabajar en el directorio API.
- D) Crear un skill que referencie los ejemplos de endpoints y contenga instrucciones de seguimiento de patrones, invocado bajo demanda mediante un comando slash. **[CORRECTA]**

Por qué D: Un skill invocado bajo demanda carga el contexto de ejemplos solo al generar nuevos endpoints, no durante tareas no relacionadas como depuración o revisión. Esto mantiene el contexto principal limpio mientras preserva la generación de alta calidad cuando se necesita.

Pregunta 39 (Escenario: Generación de código con Claude Code)

Situación: Tu equipo creó un skill `/migration` que genera archivos de migración de base de datos. Toma el nombre de la migración vía `$ARGUMENTS`. En producción observas tres problemas: (1) los desarrolladores a menudo ejecutan el skill sin argumentos, causando archivos mal nombrados, (2) el skill a veces usa detalles de esquema de base de datos de conversaciones previas no relacionadas, y (3) un desarrollador ejecutó accidentalmente limpieza de pruebas destructiva cuando el skill tenía amplio acceso a herramientas.

¿Qué enfoque de configuración corrige los tres problemas?

- A) Usar parámetros posicionales `$1` y `$2` en lugar de `$ARGUMENTS` para imponer entradas específicas, incluir referencias explícitas al archivo de esquema vía sintaxis `@` para control de

contexto, y agregar una descripción en frontmatter advirtiendo sobre operaciones destructivas.

- B) Agregar `argument-hint` en el frontmatter para solicitar parámetros requeridos, usar `context: fork` para aislar la ejecución y restringir `allowed-tools` a operaciones de escritura de archivos. **[CORRECTA]**
- C) Dividir en skills `/migration-create` y `/migration-apply`, agregar instrucciones de validación para solicitar el nombre de migración si falta, y usar diferentes alcances de `allowed-tools` para cada uno.
- D) Agregar instrucciones de validación en el SKILL.md del skill para asegurar que `$ARGUMENTS` sea un nombre válido, agregar prompts para ignorar el contexto de conversación previa, y listar operaciones prohibidas a evitar.

Por qué B: Esto usa tres características de configuración separadas para abordar cada problema: `argument-hint` mejora la entrada de argumentos y reduce los argumentos faltantes, `context: fork` previene fugas de contexto de conversaciones previas, y `allowed-tools` limita el skill a operaciones seguras de escritura de archivos, previniendo acciones destructivas.

Pregunta 40 (Escenario: Generación de código con Claude Code)

Situación: Tu base de código contiene áreas con diferentes convenciones de codificación: los componentes React usan estilo funcional con hooks, los manejadores de API usan async/await con manejo específico de errores, y los modelos de base de datos siguen el patrón repository. Los archivos de prueba están distribuidos por la base de código junto al código bajo prueba (por ejemplo, `Button.test.tsx` junto a `Button.tsx`), y quieres que todas las pruebas sigan las mismas convenciones independientemente de la ubicación.

¿Cuál es la forma más soportada de asegurar que Claude aplique automáticamente las convenciones correctas al generar código?

- A) Poner todas las convenciones en el CLAUDE.md raíz bajo encabezados para cada área y confiar en que Claude infiera qué sección aplica.
- B) Crear skills en `.claude/skills/` para cada tipo de código, incrustando convenciones en cada SKILL.md.
- C) Colocar un archivo CLAUDE.md separado en cada subdirectorio que contenga las convenciones para esa área.
- D) Crear archivos de regla bajo `.claude/rules/` con frontmatter YAML especificando patrones glob para aplicar convenciones condicionalmente según rutas de archivo. **[CORRECTA]**

Por qué D: Los archivos `.claude/rules/` con frontmatter YAML y patrones glob (por ejemplo, `**/*.test.tsx`, `src/api/**/*.ts`) habilitan aplicación de convenciones determinista basada en rutas independiente de la estructura de directorios. Este es el enfoque más soportado para patrones transversales como archivos de prueba distribuidos.

Pregunta 41 (Escenario: Generación de código con Claude Code)

Situación: Quieres crear un comando slash personalizado `/review` que ejecute la checklist estándar de revisión de código de tu equipo. Debe estar disponible para cada desarrollador cuando clone o actualice el repositorio.

¿Dónde deberías crear el archivo del comando?

- A) En `~/.claude/commands/` en el directorio home de cada desarrollador.
- B) En el repositorio del proyecto bajo `.claude/commands/`. **[CORRECTA]**
- C) En `.claude/config.json` como un array de comandos.
- D) En el CLAUDE.md raíz del proyecto.

Por qué B: Poner los comandos slash personalizados bajo `.claude/commands/` dentro del repositorio del proyecto asegura que estén versionados y automáticamente disponibles para cada desarrollador que clone o actualice el repo. Esta es la ubicación prevista para comandos personalizados a nivel de proyecto en Claude Code.

Pregunta 42 (Escenario: Generación de código con Claude Code)

Situación: El CLAUDE.md de tu equipo creció más allá de 500 líneas mezclando convenciones de TypeScript, guía de pruebas, patrones de API y procedimientos de despliegue. Los desarrolladores encuentran difícil ubicar y actualizar las secciones correctas.

¿Qué enfoque soporta Claude Code para organizar las instrucciones a nivel de proyecto en módulos temáticos enfocados?

- A) Definir un archivo `.claude/config.yaml` mapeando patrones de archivo a secciones específicas dentro de CLAUDE.md.
- B) Crear archivos Markdown separados en `.claude/rules/`, cada uno cubriendo un tema (por ejemplo, `testing.md`, `api-conventions.md`). **[CORRECTA]**
- C) Dividir las instrucciones en archivos README.md en subdirectorios relevantes que Claude carga automáticamente como instrucciones.
- D) Crear múltiples archivos llamados CLAUDE.md en diferentes niveles del árbol de directorios, cada uno sobrescribiendo las instrucciones del padre.

Por qué B: Claude Code soporta un directorio `.claude/rules/` donde puedes crear archivos Markdown separados para guía temática (por ejemplo, `testing.md`, `api-conventions.md`), permitiendo a los equipos organizar grandes conjuntos de instrucciones en módulos enfocados y mantenibles.

Pregunta 43 (Escenario: Generación de código con Claude Code)

Situación: Creas un skill personalizado `/explore-alternatives` que tu equipo usa para hacer brainstorming y evaluar enfoques de implementación antes de elegir uno. Los desarrolladores reportan que después de ejecutar el skill, las respuestas posteriores de Claude son influenciadas por la discusión de alternativas—a veces referenciando enfoques rechazados o reteniendo contexto de exploración que interfiere con la implementación real.

¿Cómo deberías configurar este skill más efectivamente?

- A) Usar el prefijo `!` en el skill para ejecutar la lógica de exploración como un subproceso bash.
- B) Agregar `context: fork` en el frontmatter del skill. **[CORRECTA]**
- C) Dividir en dos skills—`/explore-start` y `/explore-end`—para marcar límites cuando el contexto de exploración debe ser descartado.
- D) Crear el skill en `~/.claude/skills/` en lugar de `.claude/skills/`.

Por qué B: `context: fork` ejecuta el skill en un contexto de subagente aislado para que las discusiones de exploración no contaminen el historial de conversación principal. Esto previene que enfoques rechazados y el contexto de brainstorming influyan en el trabajo de implementación posterior.

Pregunta 44 (Escenario: Generación de código con Claude Code)

Situación: Tu equipo quiere agregar un servidor MCP de GitHub para buscar PRs y verificar el estado de CI vía Claude Code. Cada uno de los seis desarrolladores tiene su propio token de acceso personal de GitHub. Quieres herramientas consistentes en el equipo sin commitear credenciales al control de versiones.

¿Qué enfoque de configuración es más efectivo?

- A) Hacer que cada desarrollador agregue el servidor en alcance de usuario vía `claude mcp add --scope user`.
- B) Crear un wrapper de servidor MCP que lea tokens de un archivo `.env` y haga proxy de las llamadas a la API de GitHub, luego agregar el wrapper al `.mcp.json` del proyecto.
- C) Agregar el servidor al `.mcp.json` del proyecto usando sustitución de variables de entorno (`${GITHUB_TOKEN}`) para autenticación y documentar la variable de entorno requerida en el README del proyecto. **[CORRECTA]**
- D) Configurar el servidor en alcance de proyecto con un token marcador de posición, luego decir a los desarrolladores que lo sobrescriban en su configuración local.

Por qué C: Un `.mcp.json` de proyecto con sustitución de variables de entorno es idiomático: proporciona una única fuente de verdad versionada para la configuración MCP mientras permite a cada desarrollador suministrar credenciales vía variables de entorno. Documentar la variable hace fácil el onboarding sin commitear secretos.

Pregunta 45 (Escenario: Generación de código con Claude Code)

Situación: Estás agregando wrappers de manejo de errores alrededor de llamadas a APIs externas en una base de código de 120 archivos. El trabajo tiene tres fases: (1) descubrir todos los sitios de llamada y patrones, (2) diseñar colaborativamente el enfoque de manejo de errores, y (3) implementar wrappers de forma consistente. En la Fase 1, Claude genera salida grande listando cientos de sitios de llamada con contexto, llenando rápidamente la ventana de contexto antes de que termine el descubrimiento.

¿Qué enfoque es más efectivo para completar la tarea manteniendo la consistencia de implementación?

- A) Usar un subagente Explore para la Fase 1 para aislar la salida verbosa de descubrimiento y devolver un resumen, luego continuar las Fases 2–3 en la conversación principal. **[CORRECTA]**
- B) Hacer todas las fases en la conversación principal, usando periódicamente `/compact` para reducir el uso de contexto mientras se avanza por los archivos.
- C) Cambiar a modo headless con `--continue`, pasando resúmenes de contexto explícitos entre llamadas en lote para mantener la continuidad.
- D) Definir el patrón de manejo de errores en CLAUDE.md, luego procesar archivos en lotes a través de múltiples sesiones confiando en el archivo de memoria compartida para la consistencia.

Por qué A: Un subagente Explore aísla la salida verbosa de descubrimiento en un contexto separado y devuelve solo un resumen conciso a la conversación principal. Esto preserva la ventana de contexto principal para las fases de diseño colaborativo e implementación consistente donde el contexto retenido es más valioso.

Escenario: Agente de soporte al cliente

Pregunta 46 (Escenario: Agente de soporte al cliente)

Situación: Mientras pruebas, notas que el agente a menudo llama a `get_customer` cuando los usuarios preguntan sobre el estado del pedido, aunque `lookup_order` sería más apropiado. ¿Qué deberías verificar primero para abordar este problema?

¿Qué deberías verificar primero?

- A) Implementar un clasificador de preprocesamiento para detectar solicitudes relacionadas con pedidos y enrutarlas directamente a `lookup_order`.
- B) Reducir el número de herramientas disponibles para el agente para simplificar la elección.
- C) Agregar ejemplos few-shot al prompt del sistema cubriendo todos los patrones posibles de solicitud de pedido para mejorar la selección de herramientas.
- D) Verificar las descripciones de las herramientas para asegurar que diferencien claramente el propósito de cada una. **[CORRECTA]**

Por qué D: Las descripciones de herramientas son la entrada principal que el modelo usa para decidir qué herramienta llamar. Cuando un agente elige consistentemente la herramienta equivocada, el primer paso de diagnóstico es verificar que las descripciones de herramientas separen claramente el propósito y los límites de uso de cada una.

Pregunta 47 (Escenario: Agente de soporte al cliente)

Situación: Tu agente maneja solicitudes de un solo problema con 94% de precisión (por ejemplo, "Necesito un reembolso para el pedido #1234"). Pero cuando los clientes incluyen múltiples problemas en un solo mensaje (por ejemplo, "Necesito un reembolso para el pedido #1234 y también quiero actualizar la dirección de envío del pedido #5678"), la precisión de selección de herramientas baja al 58%. El agente generalmente resuelve solo un problema o mezcla parámetros entre solicitudes. ¿Qué enfoque mejora más efectivamente la confiabilidad para solicitudes de múltiples problemas?

¿Qué enfoque es más efectivo?

- A) Implementar una capa de preprocesamiento que use una llamada de modelo separada para descomponer mensajes de múltiples problemas en solicitudes separadas, manejar cada una independientemente y fusionar resultados.
- B) Combinar herramientas relacionadas en menos herramientas universales.
- C) Agregar ejemplos few-shot al prompt demostrando razonamiento correcto y secuenciación de herramientas para solicitudes de múltiples problemas. **[CORRECTA]**
- D) Implementar validación de respuesta que detecte respuestas incompletas y reprompte automáticamente al agente para resolver problemas omitidos.

Por qué C: Los ejemplos few-shot que demuestran razonamiento correcto y secuenciación de herramientas para solicitudes de múltiples problemas son los más efectivos porque el agente ya funciona bien en problemas únicos—lo que necesita es guía sobre el patrón para descomponer y enrutar múltiples problemas y mantener los parámetros separados.

Pregunta 48 (Escenario: Agente de soporte al cliente)

Situación: Los registros de producción muestran que para solicitudes simples como "reembolso para el pedido #1234", tu agente resuelve el problema en 3–4 llamadas a herramientas con 91% de éxito. Pero para solicitudes complejas como "Me cobraron dos veces, mi descuento no se aplicó y quiero cancelar", el agente promedia 12+ llamadas a herramientas con solo 54% de éxito—a menudo investigando problemas secuencialmente y obteniendo datos de cliente redundantes para cada uno. ¿Qué cambio mejora más efectivamente el manejo de solicitudes complejas?

¿Qué cambio es más efectivo?

- A) Agregar puntos de control explícitos de verificación entre etapas, requiriendo que el agente registre el progreso después de resolver cada problema antes de pasar al siguiente.
- B) Reducir el número de herramientas combinando `get_customer`, `lookup_order` y herramientas relacionadas con facturación en una sola herramienta `investigate_issue`.

- C) Descomponer la solicitud en problemas separados, luego investigar cada uno en paralelo usando contexto de cliente compartido antes de sintetizar una resolución final. **[CORRECTA]**
- D) Agregar ejemplos few-shot al prompt del sistema demostrando secuencias ideales de llamadas a herramientas para varios escenarios de facturación multifacéticos.

Por qué C: Descomponer en problemas separados e investigar en paralelo con contexto de cliente compartido corrige ambos problemas clave: elimina la recuperación de datos redundante reutilizando el contexto compartido entre problemas y reduce los bucles totales de llamadas a herramientas paralelizando la investigación antes de sintetizar una sola resolución.

Pregunta 49 (Escenario: Agente de soporte al cliente)

Situación: Tu agente logra 55% de resolución en primer contacto, muy por debajo del objetivo del 80%. Los registros muestran que escala casos simples (reemplazos estándar para mercancía dañada con prueba fotográfica) mientras intenta manejar autónomamente situaciones complejas que requieren excepciones de política. ¿Cuál es la forma más efectiva de mejorar la calibración de escalación?

¿Cuál es la forma más efectiva de mejorar la calibración de escalación?

- A) Requerir que el agente se autocalifique en confianza en una escala de 1–10 antes de cada respuesta y enrute automáticamente a humanos cuando la confianza caiga por debajo de un umbral.
- B) Desplegar un modelo clasificador separado entrenado en tickets históricos para predecir qué solicitudes necesitan escalación antes de que el agente principal comience a procesarlas.
- C) Agregar criterios de escalación explícitos al prompt del sistema con ejemplos few-shot que muestren cuándo escalar versus resolver autónomamente. **[CORRECTA]**
- D) Implementar análisis de sentimiento para determinar el nivel de frustración del cliente y escalar automáticamente más allá de un umbral de sentimiento negativo.

Por qué C: Los criterios de escalación explícitos con ejemplos few-shot abordan directamente la causa raíz—límites de decisión poco claros entre casos simples y complejos. Es la primera intervención más proporcional y efectiva que enseña al agente cuándo escalar y cuándo resolver autónomamente sin infraestructura adicional.

Pregunta 50 (Escenario: Agente de soporte al cliente)

Situación: Después de llamar a `get_customer` y `lookup_order`, el agente tiene todos los datos disponibles del sistema pero aún enfrenta incertidumbre. ¿Qué situación es el disparador más justificado para llamar a `escalate_to_human`?

¿Qué situación es la más justificada para escalación?

- A) Un cliente quiere cancelar un pedido enviado ayer y que llega mañana. El agente debería escalar porque el cliente podría cambiar de opinión después de recibir el paquete.
- B) Un cliente afirma que no recibió un pedido, pero el seguimiento muestra que fue entregado y firmado en su dirección hace tres días. El agente debería escalar porque presentar evidencia contradictoria podría dañar la relación con el cliente.

- C) Un cliente solicita igualar el precio de un competidor. Tus políticas permiten ajustes de precio para bajadas de precio en tu propio sitio dentro de 14 días, pero no dicen nada sobre precios de competidores. El agente debería escalar para interpretación de política. **[CORRECTA]**
- D) Un mensaje de cliente contiene tanto una pregunta de facturación como una devolución de producto. El agente debería escalar para que un humano coordine ambos problemas en una sola interacción.

Por qué C: Esta es una verdadera laguna de política: las reglas de la empresa cubren bajadas de precio en tu propio sitio pero no abordan la igualación de precios de competidores. El agente no debe inventar política y debería escalar para juicio humano sobre cómo interpretar o extender las reglas existentes.

Pregunta 51 (Escenario: Agente de soporte al cliente)

Situación: Los registros de producción muestran que en el 12% de los casos tu agente omite `get_customer` y llama a `lookup_order` directamente usando solo el nombre proporcionado por el cliente, a veces llevando a cuentas mal identificadas y reembolsos incorrectos. ¿Qué cambio corrige más efectivamente este problema de confiabilidad?

¿Qué cambio es más efectivo?

- A) Agregar ejemplos few-shot que muestren que el agente siempre llama primero a `get_customer`, incluso cuando los clientes proporcionan voluntariamente detalles del pedido.
- B) Implementar un clasificador de enrutamiento que analice cada solicitud y habilite solo un subconjunto de herramientas apropiadas para ese tipo de solicitud.
- C) Agregar una precondición programática que bloquee `lookup_order` y `process_refund` hasta que `get_customer` devuelva un identificador de cliente verificado. **[CORRECTA]**
- D) Fortalecer el prompt del sistema indicando que la verificación del cliente vía `get_customer` es obligatoria antes de cualquier operación de pedido.

Por qué C: Una precondición programática proporciona una garantía determinista de que se sigue la secuenciación requerida. Es el enfoque más efectivo porque elimina la posibilidad de saltarse la verificación, independientemente del comportamiento del LLM.

Pregunta 52 (Escenario: Agente de soporte al cliente)

Situación: Las métricas de producción muestran que al resolver disputas complejas de facturación o devoluciones de múltiples pedidos, los puntajes de satisfacción del cliente son 15% más bajos que para casos simples—incluso cuando la resolución es técnicamente correcta. El análisis de causa raíz muestra que el agente proporciona soluciones precisas pero explica la justificación de forma inconsistente: a veces omitiendo detalles de política relevantes, a veces perdiendo información de cronograma o próximos pasos. Las brechas de contexto específicas varían caso por caso. Quieres mejorar la calidad de las soluciones sin agregar supervisión humana. ¿Qué enfoque es más efectivo?

¿Qué enfoque es más efectivo?

- A) Agregar una etapa de autocrítica donde el agente evalúe un borrador de respuesta para completitud —asegurando que resuelva el problema del cliente, incluya contexto relevante y anticipe preguntas de seguimiento. **[CORRECTA]**
- B) Agregar una etapa de confirmación donde el agente pregunte "¿Esto resuelve completamente tu problema?" antes de cerrar, permitiendo a los clientes solicitar información adicional si la necesitan.
- C) Actualizar el modelo de Haiku a Sonnet para casos complejos, enrutando según una métrica de complejidad definida.
- D) Implementar ejemplos few-shot en el prompt del sistema que muestren explicaciones completas para cinco tipos comunes de casos complejos, demostrando cómo incluir contexto de política, cronogramas y próximos pasos.

Por qué A: Una etapa de autocrítica (el patrón evaluador-optimizador) aborda directamente la inconsistencia en completitud de explicación al forzar al agente a evaluar su propio borrador contra criterios concretos—como contexto de política, cronogramas y próximos pasos—antes de presentarlo. Esto detecta brechas específicas de caso sin supervisión humana.

Pregunta 53 (Escenario: Agente de soporte al cliente)

Situación: Las métricas de producción muestran que tu agente promedia 4+ bucles de API por resolución. El análisis revela que Claude a menudo solicita `get_customer` y `lookup_order` en turnos secuenciales separados incluso cuando ambos se necesitan inicialmente. ¿Cuál es la forma más efectiva de reducir el número de bucles?

¿Cuál es la forma más efectiva de reducir bucles?

- A) Implementar ejecución especulativa que llame automáticamente a herramientas probablemente necesarias en paralelo con cualquier herramienta solicitada y devuelva todos los resultados independientemente de lo que se solicitó.
- B) Aumentar `max_tokens` para dar a Claude más espacio para planificar y combinar naturalmente solicitudes de herramientas.
- C) Crear herramientas compuestas como `get_customer_with_orders` que agrupen combinaciones comunes de búsqueda en llamadas únicas.
- D) Instruir a Claude en el prompt para agrupar solicitudes de herramientas en un turno y devolver todos los resultados juntos antes de la siguiente llamada a la API. **[CORRECTA]**

Por qué D: Promptear a Claude para agrupar solicitudes de herramientas relacionadas en un solo turno aprovecha su capacidad nativa de solicitar múltiples herramientas a la vez. Corrige directamente el patrón de llamada secuencial con cambio arquitectónico mínimo.

Pregunta 54 (Escenario: Agente de soporte al cliente)

Situación: Los registros de producción muestran un patrón: los clientes referencian montos específicos (por ejemplo, "el descuento del 15% que mencioné"), pero el agente responde con valores incorrectos. La investigación muestra que estos detalles fueron mencionados 20+ turnos atrás y condensados en resúmenes vagos como "se discutieron precios promocionales". ¿Qué corrección es más efectiva?

¿Qué corrección es más efectiva?

- A) Aumentar el umbral de resumición del 70% al 85% para que las conversaciones tengan más espacio antes de que se dispare la resumición.
- B) Almacenar el historial completo de conversación en almacenamiento externo e implementar recuperación cuando el agente detecta referencias como "como mencioné".
- C) Extraer hechos transaccionales (montos, fechas, números de pedido) en un bloque persistente de "hechos del caso" incluido en cada prompt fuera del historial resumido. **[CORRECTA]**
- D) Revisar el prompt de resumición para preservar explícitamente todos los números, porcentajes, fechas y expectativas declaradas por el cliente literalmente.

Por qué C: La resumición pierde inherentemente detalles precisos. Extraer hechos transaccionales en un bloque estructurado de "hechos del caso" fuera del historial resumido preserva información crítica para que esté disponible confiablemente en cada prompt independientemente de cuántos turnos hayan sido resumidos.

Pregunta 55 (Escenario: Agente de soporte al cliente)

Situación: Tu herramienta `get_customer` devuelve todas las coincidencias al buscar por nombre. Actualmente, cuando hay múltiples resultados, Claude elige al cliente con el pedido más reciente, pero los datos de producción muestran que esto selecciona la cuenta equivocada el 15% del tiempo para coincidencias ambiguas. ¿Cómo deberías abordar esto?

¿Cómo deberías abordar esto?

- A) Implementar un sistema de puntuación de confianza que actúe autónomamente por encima del 85% de confianza y solicite aclaración por debajo del umbral.
- B) Instruir a Claude para solicitar un identificador adicional (email, teléfono o número de pedido) cuando `get_customer` devuelva múltiples coincidencias antes de tomar cualquier acción específica del cliente. **[CORRECTA]**
- C) Modificar `get_customer` para devolver solo una sola coincidencia más probable basada en un algoritmo de ranking, eliminando la ambigüedad.
- D) Agregar ejemplos few-shot al prompt demostrando razonamiento correcto y secuenciación de herramientas para coincidencias ambiguas.

Por qué B: Pedir al usuario un identificador adicional es la forma más confiable de resolver ambigüedad porque el usuario tiene conocimiento definitivo de su identidad. Un turno conversacional extra es un precio pequeño a pagar para eliminar una tasa de error del 15% causada por elegir la cuenta equivocada.

Pregunta 56 (Escenario: Agente de soporte al cliente)

Situación: Los registros de producción muestran un patrón consistente: cuando los clientes incluyen la palabra "cuenta" en su mensaje (por ejemplo, "Quiero verificar mi cuenta por un pedido que hice ayer"), el agente llama a `get_customer` primero el 78% del tiempo. Cuando los clientes formulan solicitudes similares sin "cuenta" (por ejemplo, "Quiero verificar un pedido que hice ayer"), llama a `lookup_order`

primero el 93% del tiempo. Las descripciones de herramientas son claras e inequívocas. ¿Cuál es la causa raíz más probable de esta discrepancia?

¿Cuál es la causa raíz más probable?

- A) El prompt del sistema contiene instrucciones sensibles a palabras clave que dirigen el comportamiento basadas en términos como "cuenta", creando patrones no intencionados de selección de herramientas. **[CORRECTA]**
- B) El entrenamiento base del modelo crea asociaciones entre la terminología "cuenta" y operaciones relacionadas con clientes que sobrescriben las descripciones de herramientas.
- C) El modelo necesita más datos de entrenamiento sobre mensajes multi-concepto y debería ser ajustado en ejemplos que contengan terminología de cuenta y pedido.
- D) Las descripciones de herramientas necesitan ejemplos negativos adicionales que especifiquen cuándo NO usar cada herramienta para prevenir esta confusión inducida por palabras clave.

Por qué A: El patrón sistemático impulsado por palabras clave (78% vs 93%) indica fuertemente lógica de enrutamiento explícita en el prompt del sistema reaccionando a la palabra "cuenta" y dirigiendo al agente hacia herramientas relacionadas con clientes. Como las descripciones de herramientas ya son claras, la discrepancia apunta a instrucciones a nivel de prompt creando dirección de comportamiento no intencionada.

Pregunta 57 (Escenario: Agente de soporte al cliente)

Situación: Los registros de producción muestran que el agente a menudo llama a `get_customer` cuando los usuarios preguntan sobre pedidos (por ejemplo, "verifica mi pedido #12345") en lugar de llamar a `lookup_order`. Ambas herramientas tienen descripciones mínimas ("Obtiene información del cliente" / "Obtiene detalles del pedido") y aceptan formatos de identificadores de aspecto similar. ¿Cuál es el primer paso más efectivo para mejorar la confiabilidad de selección de herramientas?

¿Cuál es el primer paso más efectivo?

- A) Implementar una capa de enrutamiento que analice la entrada del usuario antes de cada turno y preseleccione la herramienta correcta basada en palabras clave detectadas y patrones de ID.
- B) Combinar ambas herramientas en una sola `lookup_entity` que acepte cualquier identificador y decida internamente qué backend consultar.
- C) Agregar ejemplos few-shot al prompt del sistema demostrando patrones correctos de selección de herramientas, con 5–8 ejemplos enrutando consultas relacionadas con pedidos a `lookup_order`.
- D) Expandir la descripción de cada herramienta para incluir formatos de entrada, consultas de ejemplo, casos límite y límites explicando cuándo usarla versus herramientas similares. **[CORRECTA]**

Por qué D: Expandir las descripciones de herramientas con formatos de entrada, consultas de ejemplo, casos límite y límites claros corrige directamente la causa raíz—descripciones mínimas que no dan al LLM suficiente información para distinguir herramientas similares. Es un primer paso de bajo esfuerzo y alto impacto que mejora el mecanismo principal que el LLM usa para selección de herramientas.

Pregunta 58 (Escenario: Agente de soporte al cliente)

Situación: Estás implementando el bucle del agente para tu agente de soporte. Después de cada llamada a la API de Claude, debes decidir si continuar el bucle (ejecutar las herramientas solicitadas y llamar a Claude de nuevo) o detenerte (presentar la respuesta final al cliente). ¿Qué determina esta decisión?

¿Qué determina esta decisión?

- A) Verificar el campo `stop_reason` en la respuesta de Claude—continuar si es `tool_use` y detenerse si es `end_turn`. **[CORRECTA]**
- B) Parsear el texto de Claude para frases como "He terminado" o "¿Puedo ayudarte con algo más?"—las señales de lenguaje natural indican finalización de tarea.
- C) Establecer un conteo máximo de iteraciones (por ejemplo, 10 llamadas) y detenerse cuando se alcance, independientemente de si Claude indica que se necesita más trabajo.
- D) Verificar si la respuesta contiene contenido de texto del asistente—si Claude generó texto explicativo, el bucle debería terminar.

Por qué A: `stop_reason` es la señal estructurada explícita de Claude para el control del bucle: `tool_use` indica que Claude quiere ejecutar una herramienta y recibir resultados de vuelta, mientras que `end_turn` indica que Claude ha completado su respuesta y el bucle debería terminar.

Pregunta 59 (Escenario: Agente de soporte al cliente)

Situación: Los registros de producción muestran que el agente malinterpreta salidas de tus herramientas MCP: marcas de tiempo Unix de `get_customer`, fechas ISO 8601 de `lookup_order` y códigos de estado numéricos (1=pendiente, 2=enviado). Algunas herramientas son servidores MCP de terceros que no puedes modificar. ¿Qué enfoque para normalización de formato de datos es más mantenible?

¿Qué enfoque es más mantenible?

- A) Usar un hook PostToolUse para interceptar las salidas de herramientas y aplicar transformaciones de formato antes de que el agente las procese. **[CORRECTA]**
- B) Modificar las herramientas que controlas para que devuelvan formatos legibles por humanos y crear wrappers para las herramientas de terceros.
- C) Crear una herramienta `normalize_data` que el agente llame después de cada recuperación de datos para transformar valores.
- D) Agregar documentación detallada de formato al prompt del sistema explicando las convenciones de datos de cada herramienta.

Por qué A: Un hook PostToolUse proporciona un punto centralizado y determinista para interceptar y normalizar todas las salidas de herramientas—incluyendo datos de servidores MCP de terceros—antes de que el agente las procese. Es más mantenible porque las transformaciones viven en el código y aplican uniformemente, en lugar de depender de la interpretación del LLM.

Pregunta 60 (Escenario: Agente de soporte al cliente)

Situación: Los registros de producción muestran que el agente a veces elige `get_customer` cuando `lookup_order` sería más apropiado, especialmente para consultas ambiguas como "Necesito ayuda con mi compra reciente". Decides agregar ejemplos few-shot al prompt del sistema para mejorar la selección de herramientas. ¿Qué enfoque aborda más efectivamente el problema?

¿Qué enfoque es más efectivo?

- A) Agregar guía explícita de "usar cuando" y "no usar cuando" en cada descripción de herramienta cubriendo casos ambiguos.
- B) Agregar ejemplos agrupados por herramienta—todos los escenarios de `get_customer` juntos, luego todos los escenarios de `lookup_order`.
- C) Agregar 4–6 ejemplos dirigidos a escenarios ambiguos, cada uno con justificación de por qué se eligió una herramienta sobre alternativas plausibles. **[CORRECTA]**
- D) Agregar 10–15 ejemplos de solicitudes claras e inequívocas demostrando la elección correcta de herramienta para escenarios típicos para cada herramienta.

Por qué C: Dirigir los ejemplos few-shot a los escenarios ambiguos específicos donde ocurren los errores, con justificación explícita de por qué una herramienta es preferible a las alternativas, enseña al modelo el proceso comparativo de decisión necesario para casos límite. Esto es más efectivo que ejemplos genéricos o reglas declarativas.

Escenario: Patrones de arquitectura de IA conversacional

Pregunta 61 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: Tu herramienta `remove_team_member` usa un parámetro `dry_run: boolean` para previsualizar impactos antes de ejecutar. El monitoreo de producción muestra que el agente omite el paso de previsualización y llama directamente con `dry_run=false`. Necesitas garantizar que cada eliminación esté precedida por una previsualización que el usuario confirme explícitamente.

¿Cuál es el enfoque más confiable?

- A) Agregar validación en el servidor que permita `dry_run=false` solo cuando una llamada con `dry_run=true` con parámetros idénticos ocurrió en los últimos 60 segundos.
- B) Anotar la herramienta como que requiere confirmación y configurar la capa de orquestación para solicitar aprobación del usuario antes de reenviar llamadas a herramientas anotadas.
- C) Agregar instrucciones detalladas y ejemplos few-shot en la descripción de la herramienta exigiendo que el agente siempre llame primero con `dry_run=true` y espere la confirmación del usuario.
- D) Reemplazar con dos herramientas: `preview_remove_member` devuelve detalles del impacto y un token de confirmación de uso único; `execute_remove_member` requiere ese token, vinculando la ejecución a la previsualización. **[CORRECTA]**

Por qué D: El enfoque de vinculación por token hace arquitectónicamente imposible ejecutar sin una previsualización previa. La herramienta de ejecución literalmente requiere un token que solo la herramienta de previsualización puede generar. Es el único enfoque que aplica la restricción a nivel de código, no dependiendo del cumplimiento de instrucciones por el LLM (C), heurísticas de tiempo (A) o infraestructura de orquestación (B).

Pregunta 62 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: El monitoreo de producción muestra que tu herramienta `search_catalog` falla el 12% del tiempo: el 8% son tiempos de espera de red que tienen éxito al reintentarse, y el 4% son errores de sintaxis de consulta que nunca tienen éxito. Actualmente ambos tipos de error se devuelven de forma idéntica, causando reintentos desperdiciados.

¿Cómo deberías modificar el manejo de errores de la herramienta?

- A) Agregar ejemplos few-shot al prompt del sistema demostrando cómo distinguir errores de red de errores de sintaxis.
- B) Aplicar lógica de reintento con retroceso exponencial uniformemente a todos los errores.
- C) Implementar reintento automático con retroceso para tiempos de espera de red dentro de la herramienta; devolver errores de sintaxis inmediatamente con detalles de validación de parámetros. **[CORRECTA]**
- D) Devolver todos los errores con un indicador booleano `retryable` y detalles del tipo de error.

Por qué C: Manejar reintentos a nivel de la herramienta para errores transitorios es la abstracción correcta—la herramienta tiene conocimiento definitivo del tipo de error y puede implementar lógica de reintento determinista sin depender del agente para interpretar un indicador (D) o seguir instrucciones del prompt (A). El retroceso uniforme (B) desperdicia tiempo en errores de sintaxis que nunca tendrán éxito.

Pregunta 63 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: En varios turnos discutiendo estrategia de inversión, un usuario declaró "Tengo una tolerancia al riesgo muy baja" y luego "Quiero maximizar mis retornos." Ahora pregunta: "¿En qué debería invertir?"

¿Qué enfoque garantiza mejor que la recomendación se alinee con la prioridad real del usuario?

- A) Sacar a la luz la contradicción y pedir al usuario que aclare qué importa más. **[CORRECTA]**
- B) Proporcionar recomendaciones separadas para ambos escenarios.
- C) Proceder con la preferencia declarada más recientemente.
- D) Recomendar una cartera equilibrada sin abordar el conflicto.

Por qué A: Cuando las preferencias del usuario se contradicen directamente, sacar a la luz el conflicto y pedir aclaración es la única forma de garantizar que la recomendación se alinee con la verdadera

intención del usuario. Maximizar retornos y tolerancia al riesgo baja son objetivos fundamentalmente incompatibles que requieren una decisión humana.

Pregunta 64 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: Los usuarios refinan preferencias de playlist a lo largo de múltiples turnos. Dos mensajes después de que un usuario dijo "Me encanta el jazz," Claude pregunta "¿Qué géneros disfrutas?"

¿Cuál es la causa más probable?

- A) Claude requiere una conexión a base de datos vectorial para mantener memoria de conversación.
- B) La ventana de contexto del modelo ha sido excedida.
- C) La API de Claude requiere un parámetro `session_id`.
- D) Tu aplicación no está incluyendo mensajes anteriores en el array `messages`. **[CORRECTA]**

Por qué D: Claude no tiene memoria del lado del servidor—cada llamada a la API es sin estado. Sin incluir el historial completo de conversación en el array `messages` de cada solicitud, Claude no tiene conocimiento de turnos anteriores. Las bases de datos vectoriales (A) y `session_id` (C) no son parte de la arquitectura de Claude; el desbordamiento de ventana de contexto (B) es imposible para intercambios de dos mensajes.

Pregunta 65 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: Después de una sesión de cocina de 40 minutos, la conversación alcanza 78,000 tokens. El historial incluye alergias, escalado de recetas, términos de cocina aclarados y discusión general. Debes reducir tokens preservando información importante.

¿Qué enfoque equilibra mejor la preservación con la reducción de tokens?

- A) Resumir todo el historial de conversación.
- B) Conservar solo los 20,000 tokens más recientes.
- C) Extraer datos estructurados críticos (alergias, cantidades, preferencias), resumir la discusión general y mantener los intercambios recientes literalmente. **[CORRECTA]**
- D) Almacenar la conversación completa externamente y recuperar partes relevantes mediante búsqueda semántica.

Por qué C: El enfoque híbrido preserva la información de mayor valor al menor costo. Los hechos críticos como alergias y cantidades de recetas se extraen en un bloque estructurado compacto (previniendo la pérdida de precisión que ocurre durante la resumición), la discusión general se resume, y los intercambios recientes se mantienen literalmente para la coherencia conversacional. Las opciones A y B arriesgan perder información dietética crítica; D es excesiva para una sola sesión de cocina.

Pregunta 66 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: Los usuarios reportan que durante conversaciones extendidas el asistente pierde el rastro de temas y preferencias anteriores. Tu implementación actual conserva solo los últimos 25 pares de mensajes.

¿Cuál es la solución más efectiva?

- A) Enfoque híbrido: resumir mensajes más antiguos mientras se mantienen los recientes literalmente. **[CORRECTA]**
- B) Búsqueda de similitud vectorial sobre el historial completo de conversación.
- C) Aumentar la ventana a 50 pares de mensajes.
- D) Resumir mensajes eliminados en cada turno y anteponer el resumen acumulado.

Por qué A: El enfoque híbrido aborda ambas dimensiones del problema: retener contexto reciente exacto (crítico para la coherencia conversacional) mientras se mantiene una representación comprimida de preferencias anteriores. Aumentar la ventana (C) simplemente retrasa el mismo problema. La búsqueda vectorial (B) puede perder contexto importante que no es semánticamente similar a la consulta actual.

Pregunta 67 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: Los usuarios reportan que la latencia aumenta y los costos suben cuando las conversaciones superan los 50 turnos.

¿Cuál es la causa principal?

- A) Todo el historial de conversación se incluye con cada solicitud a la API. **[CORRECTA]**
- B) El modelo genera respuestas progresivamente más largas.
- C) Las operaciones de base de datos se ralentizan a medida que crece el historial.
- D) El modelo construye un perfil de usuario interno que requiere más procesamiento.

Por qué A: La API de Claude es completamente sin estado—cada solicitud debe incluir el historial completo de conversación en el array `messages`. A medida que las conversaciones crecen, cada solicitud lleva más tokens, lo que aumenta directamente tanto la latencia de procesamiento como el costo. El modelo no mantiene ningún estado interno entre llamadas (D es falso).

Pregunta 68 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: Después de tres meses de sesiones semanales, el historial de conversación crece a 85,000 tokens. Cuando un usuario pregunta "¿Qué concluimos sobre el tema del aislamiento?", el asistente da respuestas genéricas en lugar de referenciar discusiones anteriores.

¿Cuál es el enfoque más efectivo?

- A) Truncamiento de ventana deslizante.
- B) Resumición progresiva capturando conclusiones clave.
- C) Embeddings semánticos con recuperación de intercambios relevantes. **[CORRECTA]**
- D) Agregar etiquetas XML estructuradas marcando conclusiones de discusión.

Por qué C: La búsqueda semántica sobre el historial de conversación es el único enfoque que escala a tres meses de discusión mientras puede sacar a la luz intercambios relevantes específicos a demanda. El truncamiento deslizante (A) descartaría la mayoría del historial. La resumición progresiva (B) comprime las discusiones en abstracciones que pierden las conclusiones específicas que los usuarios buscan.

Pregunta 69 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: Durante las pruebas de QA, Claude sigue las pautas del prompt del sistema durante los primeros 10–15 turnos, pero las respuestas posteriores se desvían. La conversación sigue dentro de los límites de tokens.

¿Cuál es la mejor solución?

- A) Mover las pautas de comportamiento al primer mensaje del usuario.
- B) Iniciar una nueva conversación después de 20 turnos.
- C) Insertar mensajes de rol de usuario reforzando las pautas en puntos de interrupción de la conversación. **[CORRECTA]**
- D) Usar validación post-respuesta para regenerar respuestas no conformes.

Por qué C: La inyección periódica de recordatorios de comportamiento combate directamente la deriva de instrucciones reestableciendo restricciones a intervalos regulares a medida que se acumula el historial. Mover las pautas al primer mensaje del usuario (A) reduce su autoridad. La validación post-respuesta (D) es correctiva en lugar de preventiva y añade latencia significativa.

Pregunta 70 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: Tu tutor de IA tiene un prompt del sistema de 2,800 tokens que define metodología de enseñanza y reglas de adaptación. Después de 12 turnos, el asistente comienza a ignorar los niveles de competencia.

¿Cuál es la corrección más efectiva?

- A) Inyectar recordatorios cada 4–5 turnos.
- B) Reemplazar reglas verbosas con ejemplos few-shot que demuestren adaptación por nivel de competencia. **[CORRECTA]**
- C) Colocar las reglas críticas al final del prompt del sistema.

- D) Evaluar respuestas y regenerar si el nivel de dificultad no coincide.

Por qué B: Un prompt del sistema de 2,800 tokens con reglas declarativas es vulnerable a la deriva porque las reglas abstractas requieren que el modelo razone sobre ellas en cada turno. Reemplazar reglas verbosas con ejemplos few-shot concretos que demuestran la adaptación correcta por nivel de competencia da al modelo patrones de comportamiento claros para seguir—esto se cumple más confiablemente a lo largo de muchos turnos que las instrucciones abstractas.

Pregunta 71 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: Tu asistente debe mantener un tono entusiasta, explicar su razonamiento y hacer preguntas aclaratorias. ¿Dónde deben definirse estas pautas de comportamiento?

¿Dónde deben definirse estas pautas de comportamiento?

- A) Antepuestas a cada mensaje del usuario.
- B) En el prompt del sistema. **[CORRECTA]**
- C) En el primer mensaje del asistente.
- D) En variables de entorno.

Por qué B: El prompt del sistema está específicamente diseñado para restricciones y pautas de comportamiento persistentes que aplican a lo largo de toda la conversación. Anteponer a cada mensaje del usuario (A) es sobrecarga redundante. El primer mensaje del asistente (C) es poco confiable porque el modelo puede desviarse de sus propias declaraciones anteriores. Las variables de entorno (D) no tienen efecto en el comportamiento del modelo.

Pregunta 72 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: Los usuarios reportan aperturas de respuesta repetitivas como "¡Claro!" y "¡Con gusto te ayudo!"

¿Cuál es el enfoque más efectivo?

- A) Anteponer un mensaje de asistente parcial con una apertura de respuesta directa. **[CORRECTA]**
- B) Reducir la configuración de temperatura.
- C) Post-procesar respuestas para eliminar saludos.
- D) Agregar instrucciones al prompt del sistema para evitar esas frases.

Por qué A: Prellenar la respuesta del asistente con el inicio de una respuesta directa previene los patrones de saludo a nivel de generación—el modelo continúa desde el prellenado en lugar de generar nuevas frases de apertura. Las instrucciones del prompt del sistema (D) pueden ayudar pero son menos confiables. El post-procesamiento (C) es una solución frágil. La temperatura (B) controla la aleatoriedad, no patrones de frases específicos.

Pregunta 73 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: Un webhook notifica a tu sistema que el paquete de un usuario ha sido enviado mientras el usuario está chateando activamente. Quieres que el asistente incorpore esto naturalmente en la siguiente respuesta.

¿Cuál es el mejor enfoque?

- A) Agregar el estado de envío al prompt del sistema.
- B) Enviar un mensaje sintético de usuario inmediato.
- C) Forzar al asistente a llamar a una herramienta de estado en cada turno.
- D) Anteponer la actualización de estado como prefijo al siguiente mensaje del usuario. **[CORRECTA]**

Por qué D: Prefijar la actualización de estado al siguiente mensaje del usuario inyecta contexto en tiempo real en un límite de conversación natural sin interrumpir el flujo. Modificar el prompt del sistema (A) requiere reconstruir la sesión. Un mensaje sintético de usuario (B) puede romper el flujo natural del diálogo. Forzar una llamada de herramienta en cada turno (C) es costoso cuando los eventos son raros.

Pregunta 74 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: Los usuarios frecuentemente envían solicitudes como "Reserva un lugar para la fiesta." El asistente hace 4+ preguntas aclaratorias, causando un 35% de abandono.

¿Qué enfoque mejora mejor el equilibrio?

- A) Proceder con valores predeterminados ocultos.
- B) Hacer todas las preguntas aclaratorias en un mensaje compuesto.
- C) Declarar suposiciones explícitamente y proceder invitando correcciones. **[CORRECTA]**
- D) Usar un formulario de admisión estructurado.

Por qué C: Declarar suposiciones explícitamente y proceder le da al usuario una respuesta inmediata y útil mientras preserva su capacidad de corregir suposiciones incorrectas. Los valores predeterminados ocultos (A) dejan al usuario sin saber qué se asumió. Una lista de preguntas compuesta (B) sigue demandando esfuerzo inicial del usuario. Un formulario estructurado (D) agrega más fricción, no menos.

Pregunta 75 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: Tu asistente usa un prompt del sistema con persona de contratista. Los turnos iniciales siguen las reglas, pero para el turno 7 el asistente da consejos genéricos. La longitud de conversación es solo 2,500 tokens.

¿Cuál es la causa más probable?

- A) Los prompts del sistema solo establecen el comportamiento inicial.
- B) La atención del modelo se debilita a medida que se acumulan los turnos.
- C) Las respuestas acumuladas del asistente diluyen la influencia del prompt del sistema. **[CORRECTA]**
- D) El prompt del sistema solo se envía una vez.

Por qué C: A medida que las respuestas del asistente se acumulan en el historial de conversación, la proporción de texto que refleja las restricciones de comportamiento del prompt del sistema disminuye en relación al cuerpo creciente de contenido generado por el asistente. El modelo cada vez más sigue el patrón de sus propias salidas anteriores en lugar de las instrucciones del prompt del sistema, compounding la deriva incluso en longitudes de token cortas.

Pregunta 76 (Escenario: Patrones de arquitectura de IA conversacional)

Situación: Los usuarios hacen solicitudes vagas como "¿Puedes ayudar con el informe?" El asistente responde preguntando múltiples preguntas (¿qué informe? ¿qué ayuda? ¿cuál es el plazo?), causando un 40% de abandono.

¿Cuál es la mejor solución?

- A) Hacer suposiciones razonables, declararlas explícitamente y ofrecer ajustes. **[CORRECTA]**
- B) Clasificar la ambigüedad con un modelo más pequeño antes de responder.
- C) Usar interpretaciones predefinidas sin declarar suposiciones.
- D) Limitar al asistente a una pregunta aclaratoria por turno.

Por qué A: Proceder con suposiciones declaradas razonables elimina el intercambio de ida y vuelta por completo mientras mantiene al usuario informado y en control. Las interpretaciones silenciosas predefinidas (C) confunden a los usuarios cuando la respuesta no coincide con su intención. Un límite de una pregunta (D) sigue requiriendo turnos de ida y vuelta. Un modelo de clasificación más pequeño (B) agrega latencia y complejidad de infraestructura sin resolver el problema central de UX.

Ejercicios prácticos

Ejercicio 1: Agente de soporte al cliente

Objetivo: Hooks, precondiciones, escalada estructurada, manejo errores.

Pasos:

1. Crear agente con `get_customer`, `lookup_order`, `process_refund`, `escalate_to_human`
2. Hook `PreToolUse` que bloquee `process_refund` > $500
3. Precondición: `lookup_order` después de `get_customer` exitoso
4. Simular múltiples coincidencias: solicitar IDs adicionales
5. Escalar con estructura JSON: ID cliente, resumen, acciones, cantidad, causa

Dominios: 1 (Arquitectura), 3 (Configuración), 5 (Contexto)

Ejercicio 2: Configuración Claude Code

Objetivo: CLAUDE.md jerárquico, rules path-specific, MCP.

Pasos:

1. CLAUDE.md con @path referencias
2. Archivos `.claude/rules/` con glob-patrones
3. Habilidades con `context: fork` y `allowed-tools`
4. Servidor MCP en `.mcp.json`
5. Prueba: modo planificación vs ejecución directa

Dominios: 3 (Configuración), 2 (Herramientas)

Ejercicio 3: Pipeline extracción datos

Objetivo: JSON-schemas, tool_use, ciclos validación, procesamiento por lotes.

Pasos:

1. Herramienta con JSON-schema (required/optional, enum con "other")
2. Ciclo validación con reintento
3. Ejemplos few-shot para diferentes documentos
4. Message Batches API para 100 documentos
5. Enrutamiento a humano basado confianza

Dominios: 4 (Ingeniería Prompts), 5 (Contexto)

Apéndice: Tecnologías y conceptos

Tecnología	Aspectos clave
Claude Agent SDK	AgentDefinition, ciclos agentes, `stop_reason`, hooks, `allowedTools`
MCP	Servidores, herramientas, recursos, `isError`, `.mcp.json`
Claude Code	CLAUDE.md, `.claude/rules/`, `.claude/skills/`, modo planificación
Claude API	`tool_use`, JSON-schemas, `tool_choice`, `stop_reason`
Message Batches API	50% ahorro, 24 horas, `custom_id`, sin multi-turn
Herramientas	Read, Write, Edit, Bash, Grep, Glob

Temas fuera del alcance

Los siguientes temas **NO** aparecerán en el examen:

- Fine-tuning o entrenamiento de modelos personalizados
- Autenticación, facturación o gestión de cuenta
- Despliegue de servidores MCP
- Arquitectura interna de Claude
- Constitutional AI o RLHF
- Streaming API o rate limiting
- Configuraciones de nube específicas

Recomendaciones para preparación

1. **Crea un agente con Claude Agent SDK** -- ciclo completo con llamadas herramientas y manejo errores
2. **Configura Claude Code para proyecto real** -- CLAUDE.md jerárquico con rules path-specific
3. **Diseña herramientas MCP** -- descripciones claras, errores estructurados
4. **Construye pipeline extracción datos** -- JSON-schemas, validación/reintento, batch processing
5. **Practica ingeniería prompts** -- few-shot ejemplos, criterios explícitos
6. **Estudia gestión contexto** -- scratchpad files, delegación subagentes
7. **Entiende escalada y human-in-the-loop** -- criterios, procesos
8. **Haz examen de prueba** antes del real

Esta es la guía completa de certificación Claude Certified Architect, completamente traducida del ruso al español.

El documento incluye teoría fundamental, configuración práctica, estrategias avanzadas, dominios de examen, ejemplos de preguntas, ejercicios y recomendaciones de preparación.

Cuando usar Batch API vs API sincrónica:

Tarea	API	Razón
Verificación pre-merge de PR	**Sincrónica**	El desarrollador espera resultado; 24 horas es inaceptable
Reporte nocturno de deuda técnica	**Batch**	Resultado necesario por la mañana; 50% de ahorro
Auditoría de seguridad semanal	**Batch**	Sin prisa; 50% de ahorro
Revisión de código interactiva	**Sincrónica**	Necesita respuesta inmediata

Procesamiento de 10,000 documentos	**Batch**	Procesamiento masivo; ahorro significativo

7.3 Trabajo con custom_id

```
{
  "custom_id": "doc-invoice-2024-001",
  "params": {
    "model": "claude-sonnet-4-6",
    "max_tokens": 1024,
    "messages": [{"role": "user", "content": "Extrae datos de: ..."}]
  }
}
```

`custom_id` permite:

- Vincular el resultado con el documento original
- Si falla -- reenviar **solo** los documentos fallidos
- No reprocesar documentos que ya tuvieron éxito

7.4 Manejo de fallos en lotes

1. Envío de lote de 100 documentos
2. 95 procesados exitosamente, 5 -- fallo (context limit exceeded)
3. Identificación de fallos por `custom_id`
4. Modificación (dividir documentos largos en fragmentos)
5. Reenvío solo de los 5 documentos fallidos

7.5 Cálculo de SLA

Si se necesita resultado en 30 horas, y Batch API procesa hasta 24 horas:

- Ventana de envío: 30 - 24 = **6 horas**
- Los lotes deben enviarse no después de 24 horas antes del plazo
- Para envíos frecuentes -- dividir en ventanas de 4 horas

Capítulo 8: Estrategias de descomposición de tareas

8.1 Pipelines fijos (Prompt Chaining)

Cada paso está predefinido:

```
Documento -> Extracción de metadatos -> Extracción de datos -> Validación ->
Enriquecimiento -> Salida final
```

Cuándo usar:

- Estructura de tarea predecible (revisión siempre sigue el mismo patrón)
- Todos los pasos se conocen de antemano
- Se necesita estabilidad y reproducibilidad

8.2 Descomposición adaptativa dinámica

Las subtareas se generan basándose en resultados intermedios:

```
1. "Agrega pruebas para base de código legacy"
2. -> Primero: mapear estructura (Glob, Grep)
3. -> Descubierto: 3 módulos sin pruebas, 2 con cobertura parcial
4. -> Priorización: comenzar con módulo de pagos (riesgo alto)
5. -> Durante el proceso: descubierta dependencia de API externa
6. -> Adaptación: agregar mock para API externa antes de escribir pruebas
```

Cuándo usar:

- Tareas de investigación abiertas
- Cuando el volumen de trabajo total se desconoce de antemano
- Cuando cada paso depende de resultados anteriores

8.3 Revisión de código en múltiples pasadas

Para pull requests con 10+ archivos:

```
Pasada 1 (por archivo): Análisis de auth.ts -> lista de problemas locales
Pasada 1 (por archivo): Análisis de database.ts -> lista de problemas locales
Pasada 1 (por archivo): Análisis de routes.ts -> lista de problemas locales
...
Pasada 2 (integración): Análisis de relaciones entre archivos
  -> Problemas entre archivos: tipos inconsistentes, dependencias cíclicas
```

Por qué una sola pasada para 14 archivos es malo:

- Dilución de atención: análisis detallado de unos archivos, superficial de otros

- Comentarios contradictorios: patrón marcado como problema en un archivo, aprobado en otro
- Omisión de bugs: errores obvios omitidos por sobrecarga cognitiva

---

Capítulo 9: Escalada y Human-in-the-Loop

9.1 Cuándo escalar a un humano

Desencadenantes de escalada (reglas claras):

Situación	Acción
Cliente explícitamente pide "dame un gerente"	Escalada inmediata, sin intentar resolver
Política no cubre solicitud del cliente	Escalada (ej., comparación de precios con competencia, pero política no lo cubre)
Agente no puede lograr progreso	Escalada después de intentos razonables
Operación financiera por encima del umbral	Escalada (mejor mediante hook que mediante prompt)
Múltiples coincidencias al buscar cliente	Solicitar identificadores adicionales, no adivinar

Qué NO es desencadenante confiable:

Método poco confiable	Por qué no funciona
Análisis de sentimiento	El sentimiento del cliente no correlaciona con complejidad del caso
Auto-evaluación de confianza del modelo (1-10)	El modelo ya está incorrectamente confiado en decisiones erróneas; mal calibrado
Clasificador automático	Exceso de complejidad; requiere datos de entrenamiento que puede no haber

9.2 Patrones de escalada

Escalada inmediata:

```
Cliente: "Quiero hablar con un gerente"
Agente: [llama inmediatamente escalate_to_human]
NO: "Puedo ayudarte con tu pregunta, permíteme..."
```

Escalada con intento de solución:

```
Cliente: "Mi refrigerador se rompió 2 días después de comprar"
Agente: [verifica pedido, ofrece reemplazo por garantía]
Si cliente insatisfecho -> escalada
```

Escalada matizada (acknowledge → resolve → escalate on reiteration):

```
Cliente: "¡Esto es vergonzoso, estoy muy insatisfecho!"
Agente: [reconoce decepción] "Entiendo tu decepción."
       [ofrece solución] "Puedo ofrecer reemplazo o devolución."
Cliente: "No, ¡quiero hablar con alguien!"
Agente: [cliente reitera -> escalada inmediata]
```

Principio clave: primero reconocer emoción del cliente, luego ofrecer solución concreta, solo al reiterar insistencia -- escalar. No escales en la primera expresión de descontento (no es lo mismo que solicitar gerente).

Escalada por brecha en política:

```
Cliente: "Competencia X tiene este producto 30% más barato, haz descuento"
Política: describe solo ajustes de precio para el propio sitio
Agente: [escalada -- política no cubre comparación de precios con competencia]
```

9.3 Protocolos de transferencia estructurados (Handoff)

Al escalar, el agente debe transmitir al humano un resumen estructurado:

```
{
  "customer_id": "CUST-12345",
  "customer_name": "Juan García",
  "issue_summary": "Solicitud de devolución por producto dañado",
  "order_id": "ORD-67890",
  "root_cause": "Producto llegó dañado, fotos adjuntas",
  "actions_taken": [
    "Cliente verificado mediante get_customer",
    "Pedido confirmado mediante lookup_order",
    "Ofrecido reemplazo estándar -- cliente insiste en devolución"
  ],
  "refund_amount": "$89.99",
  "recommended_action": "Aprobar devolución completa",
  "escalation_reason": "Cliente solicitó hablar con gerente"
}
```

El operador humano **no tiene acceso a la transcripción** -- solo ve este resumen. Por lo tanto debe ser completo y autosuficiente.

9.4 Calibración de confianza y control humano

Para sistemas de extracción de datos:

1. **Field-level confidence scores:** el modelo produce puntuación de confianza para cada campo extraído
2. **Calibración:** usar conjuntos de validación anotados para ajustar umbrales
3. **Ruteo:**
    - Alta confianza + precisión estable -> procesamiento automático
    - Baja confianza o fuente ambigua -> revisión humana

Muestreo estratificado aleatorio:

- Incluso para extracciones de alta confianza, verifica regularmente muestras
- Precisión agregada del 97% puede enmascarar 40% de errores en cierto tipo de documentos
- Analiza precisión por tipo de documento y por campo, no solo en total

---

Capítulo 10: Manejo de errores en sistemas multiagente

10.1 Categorías de errores

Categoría	Ejemplos	¿Reintentable?	Acción del agente
Transitorio	Timeout, 503, fallo de red	Sí	Reintentar con backoff exponencial
Validación	Formato de entrada inválido, campo obligatorio faltante	No (necesita corregir entrada)	Cambiar solicitud e intentar de nuevo
Comercial	Violación de política, límite excedido	No	Explicar al usuario, ofrecer alternativa
Permiso	Sin permisos de acceso	No	Escalar

10.2 Antipatrones en manejo de errores

Antipatrón	Problema	Enfoque correcto
Estado genérico "search unavailable"	Coordinador no puede decidir recuperación	Devolver tipo de error, solicitud, resultados parciales, alternativas
Supresión silenciosa (resultado vacío = éxito)	Coordinador piensa sin coincidencias, cuando fue fallo	Distinguir explícitamente "sin resultados" de "fallo en búsqueda"
Terminar todo proceso si un fallo	Pérdida de todos resultados parciales	Continuar con resultados parciales, anotar vacíos
Reintentos infinitos dentro de subagente	Retraso y gasto de recursos	Recuperación local (1-2 reintentos), luego propagar a coordinador

10.3 Error estructurado de subagente

```
{
  "status": "partial_failure",
  "failure_type": "timeout",
  "attempted_query": "Impacto de IA en industria musical 2024",
  "partial_results": [
    {"title": "Reporte de Generación de Música AI", "url": "...", "relevance": 0.8}
  ],
  "alternative_approaches": [
    "Intentar búsqueda más específica: 'Herramientas de composición musical AI'",
    "Usar fuente de datos alternativa"
  ],
  "coverage_impact": "No cubierto: impacto de IA en producción musical"
}
```

El coordinador obtiene toda la información para decidir:

- ¿Reintentar con solicitud modificada?
- ¿Usar resultados parciales?
- ¿Involucrar otro subagente?
- ¿Continuar sin esta sección y anotar vacío?

10.4 Anotaciones de cobertura en síntesis final

```
## Reporte: Impacto de IA en industrias creativas

### Artes visuales (COBERTURA COMPLETA)
[resultados de investigación]

### Música (COBERTURA PARCIAL -- timeout en búsqueda)
[resultados parciales]
⚠️ Nota: cobertura de esta sección limitada por timeout del agente de búsqueda.

### Literatura (COBERTURA COMPLETA)
[resultados de investigación]
```

Capítulo 11: Gestión de contexto en sistemas de producción

11.1 Extracción de hechos en bloque separado

En lugar de depender del historial de conversación (que se degrada con sumarización), extrae hechos clave en bloque estructurado:

```
=== HECHOS DEL CASO (actualizado cada nuevo hecho) ===
ID del Cliente: CUST-12345
ID del Pedido: ORD-67890
Fecha del Pedido: 2025-01-15
Monto del Pedido: $89.99
Problema: Producto dañado en entrega
Solicitud del Cliente: Devolución completa
Estado: Esperando aprobación del gerente
===
```

Este bloque se incluye en **cada** prompt, independientemente de sumarización del historial.

11.2 Trimming de resultados de herramientas

Si `lookup_order` devuelve 40+ campos, pero solo se necesitan 5 para la tarea actual:

```
# Hook PostToolUse: mantener solo campos relevantes
@hook("PostToolUse", tool="lookup_order")
def trim_order_fields(result):
    return {
        "order_id": result["order_id"],
        "status": result["status"],
        "total": result["total"],
        "items": result["items"],
        "return_eligible": result["return_eligible"]
    }
```

Esto ahorra ventana de contexto y reduce ruido.

11.3 Entrada consciente de posición

Coloca información crítica con conciencia del efecto "lost-in-the-middle":

```
[CONCLUSIONES CLAVE -- al principio]
Se encontraron 3 vulnerabilidades críticas...

[RESULTADOS DETALLADOS -- medio]
=== Archivo auth.ts ===
...
=== Archivo database.ts ===
...

[INSTRUCCIONES DE ACCIÓN -- al final]
Prioridad: corregir vulnerabilidades en auth.ts antes de merge.
```

11.4 Archivos Scratchpad

Durante investigación prolongada de base de código, el agente puede registrar hallazgos clave en archivo scratchpad:

```
# investigation-scratchpad.md
## Hallazgos clave
- Clase PaymentProcessor en src/payments/processor.ts hereda de BaseProcessor
- Método refund() se llama desde 3 lugares: OrderController, AdminPanel, CronJob
- API externa PaymentGateway tiene limit de 100 req/min
- Migración #47 agregó campo refund_reason (NOT NULL) -- 2024-12-01
```

Cuando contexto se degrada (o en nueva sesión), el agente accede a scratchpad en lugar de reinvestigar.

11.5 Delegación a subagentes para proteger contexto

```
Agente principal: "Investiga dependencias del módulo de pagos"
  -> Subagente (Explore): lee 15 archivos, traza importes
  -> Devuelve: "Módulo de pagos depende de AuthService, OrderModel, y PaymentGateway
API externa"

Agente principal: guarda 1 línea en lugar de 15 archivos en contexto
```

Capa de contexto separada (separate context layer): En sistemas multiagente, cada subagente trabaja con presupuesto de contexto limitado -- recibe solo la información necesaria para su tarea. El coordinador actúa como capa de contexto separada: agrega resultados de subagentes, mantiene estado global y distribuye contexto. Esto previene "fuga de contexto", donde un agente consume ventana con información no relevante para otros.

Presupuestos de contexto limitados para subagentes:

- Transmite al subagente contexto mínimo: tarea específica + datos necesarios
- Instruye al subagente devolver solo resultado estructurado, no datos en bruto
- Usa `allowedTools` para limitar el conjunto de herramientas del subagente -- menos herramientas = menos distracciones y menos gasto de contexto

11.6 Guardado estructurado de estado (para crash recovery)

Cada agente exporta su estado a ubicación conocida:

```
// agent-state/web-search-agent.json
{
  "status": "completed",
  "queries_executed": ["IA music 2024", "AI music composition"],
  "results_count": 12,
  "key_findings": [...],
  "coverage": ["music composition", "music production"],
  "gaps": ["music distribution", "music licensing"]
}
```

El coordinador carga manifiesto al reanudar:

```
// agent-state/manifest.json
{
  "web-search": "completed",
  "doc-analysis": "in_progress",
  "synthesis": "not_started"
}
```

Capítulo 12: Preservación de origen de información (Provenance)

12.1 Problema de pérdida de atribución

Cuando se sumariz resultados de múltiples fuentes, se pierde conexión "afirmación -> fuente":

```
❌ Incorrecto: "El mercado de IA en música se evalúa en $3.2 mil millones."
   (¿De dónde? ¿Qué fuente? ¿Qué año?)

✅ Correcto:
{
  "claim": "El mercado de IA en música se evalúa en $3.2 mil millones.",
  "source_url": "https://example.com/report",
  "source_name": "Global AI Music Report 2024",
  "publication_date": "2024-06-15",
  "confidence": 0.9
}
```

12.2 Manejo de datos en conflicto

Cuando dos fuentes dan cifras diferentes:

```
{
  "claim": "Porcentaje de música generada por IA en plataformas de streaming",
  "values": [
    {
      "value": "12%",
      "source": "Spotify Annual Report 2024",
      "date": "2024-03",
      "methodology": "Clasificación automática"
    },
    {
      "value": "8%",
      "source": "Music Industry Association Survey",
      "date": "2024-07",
      "methodology": "Encuesta a 500 sellos"
    }
  ],
  "conflict_detected": true,
  "possible_explanation": "Diferencia en metodología y período temporal"
}
```

NO selecciones arbitrariamente un valor. Preserva ambos con atribución y deja que el coordinador decida.

12.3 Inclusión de fechas para interpretación correcta

Sin fechas, diferencias temporales se interpretan como contradicciones:

```
❌ "Fuente A dice 10%, fuente B dice 15%. Contradicción."
✅ "Fuente A (2023) dice 10%, fuente B (2024) dice 15%. Probable crecimiento del 5% anual."
```

12.4 Renderizado por tipo de contenido

No conviertas todo a único formato:

- **Datos financieros** -> tablas
- **Noticias y análisis** -> prosa
- **Hallazgos técnicos** -> listas estructuradas
- **Series temporales** -> orden cronológico

Capítulo 13: Herramientas integradas de Claude Code

13.1 Referencia de selección de herramientas

Tarea	Herramienta	Ejemplo
Encontrar archivos por nombre/patrón	**Glob**	`**/*.test.tsx`, `src/components/**/*.ts`
Encontrar contenido en archivos	**Grep**	Nombre de función, mensaje de error, import
Leer archivo completo	**Read**	Cargar archivo para análisis
Escribir archivo nuevo	**Write**	Crear archivo nuevo desde cero
Edición puntual de archivo existente	**Edit**	Reemplazar fragmento específico por texto único
Ejecutar comando shell	**Bash**	git, npm, ejecutar pruebas, compilar

13.2 Estrategia de investigación incremental

No leas todos los archivos a la vez. Construye comprensión incrementalmente:

```
1. Grep: encontrar puntos de entrada (definición de función, export)
2. Read: leer archivos encontrados
3. Grep: encontrar usos (import, llamadas)
4. Read: leer archivos consumidores
5. Repetir hasta construir comprensión completa
```

13.3 Fallback: Read + Write en lugar de Edit

Cuando Edit falla por falta de coincidencia de texto único:

1. **Read** -- cargar contenido completo del archivo
2. Modificar contenido programáticamente
3. **Write** -- escribir versión actualizada

PARTE II: COMPENDIO POR DOMINIOS DEL EXAMEN

Dominio 1: Arquitectura de agentes y orquestación (27%)

1.1 Diseño de ciclos agentes para ejecución autónoma de tareas

Conocimientos clave:

- **Ciclo de vida del agente**: enviar solicitud a Claude, verificar `stop_reason` (`"tool_use"` vs `"end_turn"`), ejecutar herramientas, devolver resultados para iteración siguiente
- Los resultados de herramientas se agregan al historial de conversación para que el modelo razone sobre la siguiente acción
- **Toma de decisiones dirigida por modelo** (Claude decide qué herramienta llamar) vs árboles de decisión predefinidos

Habilidades clave:

- Implementar control de flujo: ciclo continúa con `stop_reason = "tool_use"` y finaliza con `"end_turn"`
- Agregar resultados de herramientas al contexto entre iteraciones
- **Antipatrones a evitar**: parsear texto del asistente para determinar finalización, establecer límites de iteración arbitrarios como mecanismo principal de parada

1.2 Orquestación de sistemas multiagente (coordinador-subagente)

Conocimientos clave:

- **Arquitectura Hub-and-spoke**: coordinador gestiona toda comunicación entre agentes, manejo de errores y enrutamiento de información
- Los subagentes trabajan con **contexto aislado** -- NO heredan automáticamente el historial de conversación del coordinador
- Rol del coordinador: descomposición de tareas, delegación, agregación de resultados, selección dinámica de subagentes
- Riesgo de descomposición demasiado granular de tareas por coordinador

Habilidades clave:

- Implementar coordinador que descompone tareas, genera subagentes a través de `Task`, transmite contexto explícitamente en prompts
- Agregación de resultados de múltiples subagentes, manejo de fallos parciales
- Validación de outputs de subagentes

1.3 Manejo de ciclos agentes fallidos

Conocimientos clave:

- Cuando `stop_reason` es `"max_tokens"`, la respuesta se trunca -- aumentar `max_tokens` o dividir tarea
- Recuperación de fallos de herramientas vs fallos semánticos
- Distinguir entre errores transitorios (reintentar) vs business logic errors (propagar/escalar)

Habilidades clave:

- Implementar manejo de max_tokens: detectar, aumentar límite, reintentar
- Categorizar fallos por tipo, tomar decisión de reintento vs propagación

---

Dominio 2: Diseño de herramientas e integración MCP (18%)

2.1 Diseño de esquemas de herramientas

Conocimientos clave:

- **Descripción como mecanismo principal de selección:** la descripción debe desambiguar entre herramientas similares
- Tool descriptions incluyen: qué hace, qué devuelve, formatos de entrada/ejemplos, cuándo usar vs alternativas
- Evitar descripciones idénticas o superpuestas

Habilidades clave:

- Escribir descripción clara de herramienta con ejemplos y casos límite
- Identificar cuándo descripción insuficiente causa confusión de selección

2.2 JSON Schema para herramientas

Conocimientos clave:

- Sintaxis básica: `type`, `properties`, `required`, `enum`, `null`
- Required vs optional: los campos required fuerzan al modelo a inventar si faltan
- Nullable: `"type": ["string", "null"]` para datos que pueden faltar

Habilidades clave:

- Diseñar esquemas que no fuercen fabricación de datos

2.3 MCP Servers y configuración

Conocimientos clave:

- Configuración de proyecto (`.mcp.json`) vs personal (`~/.claude.json`)
- Conexión de múltiples servidores MCP simultáneamente
- Uso de variables de entorno para secretos, no tokens en plaintext

Habilidades clave:

- Configurar servidor MCP con variables de entorno

2.4 Manejo de errores en MCP

Conocimientos clave:

- `isError: true` indica fallo de herramienta
- Errores estructurados deben incluir: categoría (transient/validation/business), reintentable, mensaje, query intentada, partial results
- Errores genéricos no ayudan coordinadores a recuperarse

Habilidades clave:

- Diseñar respuestas de error estructuradas que permitan coordinador tomar decisiones

2.5 Recursos MCP

Conocimientos clave:

- Recursos proporcionan contexto sin ejecutar acciones
- Ejemplos: catálogos, esquemas BD, documentación, resúmenes
- Previenen necesidad de llamadas exploratorias de herramientas

---

Dominio 3: Configuración y flujos de trabajo de Claude Code (20%)

3.1 Jerarquía CLAUDE.md

Conocimientos clave:

- Tres niveles: usuario (`~/.claude/CLAUDE.md`), proyecto (`.claude/CLAUDE.md`), directorio (`CLAUDE.md` en subdirs)
- El nivel más específico prevalece

- Proyecto se gestiona a través de VCS, usuario y directorio son locales

Habilidades clave:

- Decidir qué nivel colocar instrucciones para máxima efektividad

3.2 Reglas y Skills

Conocimientos clave:

- `.claude/rules/` con YAML frontmatter y campo `paths` para carga condicional
- `.claude/skills/` para comandos reutilizables invocables mediante `/nombre`
- SKILL.md frontmatter: `context: fork`, `allowed-tools`, `argument-hint`

Habilidades clave:

- Usar `paths` glob patterns para condicionalidad
- Decidir cuándo usar skill vs CLAUDE.md

3.3 Modo de planificación vs ejecución directa

Conocimientos clave:

- Plan mode: exploratorio, sin cambios, requiere aprobación
- Direct execution: cambios inmediatos
- Usar plan mode para cambios grandes, desconocidas codebases

Habilidades clave:

- Identificar cuándo usar cada modo

3.4 Claude Code CLI y CI/CD

Conocimientos clave:

- `-p` / `--print` para modo no interactivo en CI/CD
- `--output-format json --json-schema` para salida estructurada
- Usar instancias independientes para generación vs revisión

Habilidades clave:

- Escribir prompts de Claude CLI que produzcan salida estructurada para CI

Dominio 4: Ingeniería de prompts y salida estructurada (20%)

4.1 Few-shot prompting

Conocimientos clave:

- Few-shot (2-4 ejemplos) vs descripciones textuales
- Ejemplos para casos ambiguos, formatos de salida, distinción código válido/inválido
- Ejemplos para extracción de diferentes formatos de documento

Habilidades clave:

- Escribir ejemplos few-shot relevantes

4.2 Criterios explícitos vs instrucciones vagas

Conocimientos clave:

- Especificar **SOLO SI** para claridad
- Incluir ejemplos para cada criterio de severidad
- Definir negaciones (qué NO buscar)

Habilidades clave:

- Refactor instrucciones vagas en criterios explícitos

4.3 JSON Schema y validación

Conocimientos clave:

- Sintaxis correcta JSON no garantiza semántica correcta
- Validación de estructura vs semántica
- Pydantic para validación con auto-generación de schema

Habilidades clave:

- Usar Pydantic para validar y reintentar con feedback

4.4 Prompt Chaining

Conocimientos clave:

- Dividir tareas complejas en pasos secuenciales
- Evita dilución de atención

- Proporciona calidad consistente

Habilidades clave:

- Diseñar cadena de prompts para tareas multifase

4.5 Retry-with-feedback y self-correction

Conocimientos clave:

- Validar, detectar errores, reintentar con contexto de error
- Self-correction: incluir valor afirmado Y calculado
- Conflictos detectados permiten procesamiento diferente

Habilidades clave:

- Implementar ciclos de validación/reintento

Dominio 5: Gestión de contexto y confiabilidad (15%)

5.1 Problemas de ventana de contexto

Conocimientos clave:

- Lost-in-the-middle: información del medio es menos confiable
- Acumulación de resultados de herramientas: cada call agrega datos
- Sumarización progresiva: números, fechas, porcentajes se vuelven vagas

Habilidades clave:

- Estrategias para colocar información crítica
- Trimming de resultados de herramientas

5.2 Ejecución de hechos y bloque de caso

Conocimientos clave:

- Mantener bloque estructurado de hechos clave
- Actualizar con nuevos hechos
- Incluir en cada prompt independientemente de sumarización

Habilidades clave:

- Diseñar bloque de casos que se mantenga consistente

5.3 Escalada y Human-in-the-Loop

Conocimientos clave:

- Desencadenantes confiables (solicitud explícita, política no cubre) vs no confiables (análisis de sentimiento)
- Protocolos de transmisión estructurados
- Confidence scores a nivel de campo para routing

Habilidades clave:

- Implementar escalada confiable
- Diseñar transmisiones autosuficientes

5.4 Manejo de errores en sistemas multiagente

Conocimientos clave:

- Categorías: transient, validation, business, permission
- Errores estructurados vs genéricos
- Continuar con resultados parciales, anotar brechas

Habilidades clave:

- Propagar errores estructura a coordinadores
- Usar anotaciones de cobertura en síntesis

5.5 Provenance y atribución

Conocimientos clave:

- Conservar conexión afirmación -> fuente
- Manejar datos en conflicto: ambos valores + atribución
- Incluir fechas para interpretación correcta

Habilidades clave:

- Extraer claims con fuentes, URL, fechas

PARTE III: EJERCICIOS PRÁCTICOS

Ejercicio 1: Revisión de agentes y selección de herramientas

Objetivo: Agentive loops, `stop_reason`, selección de herramientas basada en descripción.

Escenario: Agente de soporte al cliente que maneja retornos y disputas de facturas usando herramientas: `get_customer`, `lookup_order`, `process_refund`, `escalate_to_human`.

Tareas:

1. Implementar ciclo agente básico: detectar `stop_reason`, ejecutar herramientas, añadir resultados
2. Problema: modelo elige `escalate_to_human` inmediatamente en lugar de `get_customer` primero. Mejorar descripciones de herramientas.
3. Simular fallo de herramienta (timeout en `lookup_order`), manejar gracefully
4. Probar con escenarios: cliente solicita devolución vs cliente pide hablar con gerente

Dominios: 1 (Arquitectura de agentes), 2 (Diseño de herramientas)

Ejercicio 2: Configuración de proyecto para Claude Code

Objetivo: CLAUDE.md, reglas con `paths`, skills, MCP.

Tareas:

1. Crear jerarquía CLAUDE.md: convenciones del proyecto que se aplican a todos
2. `.claude/rules/` con rules específicas: uno para API files, uno para tests (con glob patterns)
3. Crear skill `/review` que use `context: fork` para análisis de código
4. Configurar MCP en `.mcp.json` con variables de entorno para GitHub y Jira
5. Validar que diferentes usuarios del proyecto obtienen la misma configuración pero pueda tener personalizaciones en `~/.claude/`

Dominios: 3 (Configuración Claude Code), 2 (Integración MCP)

Ejercicio 3: Pipeline de extracción estructurada de datos

Objetivo: JSON-schemas, tool_use para salida estructurada, ciclos validación/reintento, Message Batches.

Tareas:

1. Definir herramienta de extracción con JSON-schema (required/optional, enums con "other", nullable)
2. Implementar ciclo validación: upon error -- reintentar con documento, extracción errónea, error específico
3. Escribir ejemplos few-shot para documentos con diferentes estructuras
4. Procesamiento batch: 100 documentos, manejo de fallos por `custom_id`
5. Routing a humano: confidence scores por campo, análisis por tipo de documento

Dominios: 4 (Ingeniería de prompts), 5 (Contexto y confiabilidad)

Ejercicio 4: Pipeline multiagente con síntesis

Objetivo: Orquestación de agentes, propagación de errores, provenance, síntesis.

Tareas:

1. Coordinador con 2+ subagentes: búsqueda web, análisis de documentos
2. Contexto explícito en prompts de subagentes
3. Salida estructurada de subagentes: claim, source URL, fecha, confianza
4. Simular timeout: coordinador recibe partial results, continúa
5. Conflicto de datos: dos fuentes con valores diferentes -- preservar ambos con atribución
6. Síntesis final con anotaciones de cobertura (secciones COMPLETAMENTE CUBIERTAS vs PARCIALMENTE)

Dominios: 1 (Arquitectura de agentes), 2 (Herramientas MCP), 5 (Contexto y confiabilidad)

Apéndice: Tecnologías y conceptos

Tecnología	Aspectos clave
Claude Agent SDK	AgentDefinition, ciclos agentes, `stop_reason`, hooks (PostToolUse), generación de subagentes mediante Task, `allowedTools`
Model Context Protocol (MCP)	Servidores MCP, herramientas, recursos, `isError`, descripciones de herramientas, `.mcp.json`, variables de entorno
Claude Code	Jerarquía CLAUDE.md, `.claude/rules/` con glob-patterns, `.claude/commands/`, `.claude/skills/` con SKILL.md, modo planificación, `/compact`, `--resume`, `fork_session`
Claude Code CLI	`-p` / `--print` para modo no interactivo, `--output-format json`, `--json-schema`
Claude API	`tool_use` con JSON-schemas, `tool_choice` ("auto"/"any"/fuerza), `stop_reason`, `max_tokens`, prompts del sistema
Message Batches API	Ahorro del 50%, ventana hasta 24 horas, `custom_id`, sin tool calling multi-turn
JSON Schema	Required vs optional, campos nullable, tipos enum, `"other"` + detail, modo estricto
Pydantic	Validación de estructura, validadores personalizados, auto-generación de schema, ciclos validación/reintento
Few-shot prompting	2-4 ejemplos, casos ambiguos, formatos de salida, criterios de severidad
Provenance	Conservar afirmación -> fuente, manejar conflictos, incluir fechas

Escalada	Desencadenantes confiables, transmisiones estructuradas, confidence scores por campo
Manejo de errores	Categorías de errores, errores estructurados, partial failures, anotaciones de cobertura

---

Fin de la Guía de Estudio

Esta guía proporciona una base sólida para el examen Claude Certified Architect -- Foundations. Revisa los dominios, practica con los ejercicios y asegúrate de entender los antipatrones además de los patrones correctos.

¡Buena suerte en tu examen!