# Claude認定アーキテクト — Foundations 認定資格

## 学習ガイド（公式試験ガイドに基づく）

---

## はじめに

**Claude認定アーキテクト — Foundations** 認定資格は、実際のClaudeベースのソリューションを実装する際に、適切なトレードオフの判断ができる専門家であることを証明します。試験では、Claude Code、Claude Agent SDK、Claude API、およびModel Context Protocol（MCP）に関する基礎知識を評価します。これらは、Claudeを使った本番アプリケーションを構築するための中核技術です。

試験問題は、現実の業界シナリオに基づいています。カスタマーサポート向けのエージェントシステムの構築、マルチエージェント調査パイプラインの設計、Claude CodeのCI/CDへの統合、開発者向け生産性ツールの作成、非構造化ドキュメントからの構造化データ抽出などが含まれます。

---

## 対象者

理想的な受験者は、Claudeを使った本番アプリケーションを設計・出荷する**ソリューションアーキテクト**です。以下の分野で少なくとも6ヶ月の実務経験が必要です。

- **Claude Agent SDK** — マルチエージェントオーケストレーション、サブエージェントへの委譲、ツール統合、ライフサイクルフック
- **Claude Code** — CLAUDE.md、MCPサーバー、エージェントスキル、計画モード
- **Model Context Protocol（MCP）** — バックエンド統合のためのツールとリソース
- **プロンプトエンジニアリング** — JSONスキーマ、few-shotサンプル、データ抽出テンプレート
- **コンテキストウィンドウ** — 長文ドキュメントの扱い、マルチエージェントコンテキストの受け渡し
- **CI/CDパイプライン** — 自動コードレビュー、テスト生成
- **エスカレーションと信頼性** — エラーハンドリング、人間のループへの組み込み

---

## 試験形式

| パラメータ | 値 |
|---|---|
| 問題形式 | 択一式（4択から1つを選択） |
| スコア | 100〜1000点スケール、合格点 **720点** |
| 未解答ペナルティ | なし（すべての問題に答えること！） |
| シナリオ | 8つの中から4つをランダムに選択 |

---

## 試験内容：5つのドメイン

| ドメイン | 配点比率 |
|---|---|
| 1. エージェントアーキテクチャとオーケストレーション | **27%** |
| 2. ツール設計とMCP統合 | **18%** |
| 3. Claude Codeの設定とワークフロー | **20%** |
| 4. プロンプトエンジニアリングと構造化出力 | **20%** |
| 5. コンテキスト管理と信頼性 | **15%** |

---

## 試験シナリオ

### シナリオ1：カスタマーサポートエージェント

Claude Agent SDKを使って、返品・請求トラブル・アカウント問題を処理するエージェントを構築します。エージェントはMCPツール（`get_customer`、`lookup_order`、`process_refund`、`escalate_to_human`）を使用します。目標は80%以上の初回解決率と適切なエスカレーションです。

### シナリオ2：Claude Codeによるコード生成

Claude Codeを使って開発を加速します。コード生成、リファクタリング、デバッグ、ドキュメント作成が対象です。カスタムスラッシュコマンドとCLAUDE.md設定との統合、および計画モードの使いどころを理解する必要があります。

### シナリオ3：マルチエージェント調査システム

コーディネーターエージェントが、Webリサーチ・ドキュメント分析・統合・レポート生成などの専門サブエージェントにタスクを委譲します。システムは引用を含む完全なレポートを生成する必要があります。

### シナリオ4：開発者生産性ツール

エージェントが、エンジニアの慣れないコードベースの探索、ボイラープレートコードの生成、定型タスクの自動化を支援します。組み込みツール（Read、Write、Bash、Grep、Glob）とMCPサーバーを使用します。

### シナリオ5：継続的インテグレーション向けClaude Code

自動コードレビュー、テスト生成、プルリクエストフィードバックのためにClaude CodeをCI/CDパイプラインに統合します。偽陽性を最小限に抑えるプロンプト設計が必要です。

### シナリオ6：構造化データ抽出

非構造化ドキュメントから情報を抽出し、JSONスキーマで出力を検証し、高い精度を維持するシステムです。エッジケースを正しく処理する必要があります。

### シナリオ7：会話型AIアーキテクチャパターン

コンテキストウィンドウ管理、複数ターンにわたる指示の持続、メモリ戦略、安全な実行のためのツール設計、曖昧または矛盾するユーザー入力の処理をカバーするマルチターン会話システムを設計します。

### シナリオ8：エージェンティックAIツール *（内容未掲載 — ご協力ください！）*

このシナリオは受験者から報告されていますが、本ガイドではまだカバーされていません。実際の試験でこのシナリオの問題に遭遇した場合は、GitHub Issues で共有していただければ、完全なコンテンツを追加できます。あなたの貢献が試験準備中のすべての人の役に立ちます。

---

# 公式ドキュメント

| リソース | 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 — Overview** | 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 — Subagents** | https://platform.claude.com/docs/en/agent-sdk/subagents |
| **Claude Agent SDK — Sessions** | https://platform.claude.com/docs/en/agent-sdk/sessions |
| **Model Context Protocol（MCP）** | https://modelcontextprotocol.io/ |
| **MCP — Tools** | https://modelcontextprotocol.io/docs/concepts/tools |
| **MCP — Resources** | https://modelcontextprotocol.io/docs/concepts/resources |
| **MCP — Servers** | https://modelcontextprotocol.io/docs/concepts/servers |
| **Claude Code — ドキュメント** | https://code.claude.com/docs/en/overview |
| **Claude Code — CLAUDE.mdとメモリ** | https://code.claude.com/docs/en/memory |
| **Claude Code — スキル（スラッシュコマンドを含む）** | https://code.claude.com/docs/en/skills |
| **Claude Code — Hooks** | https://code.claude.com/docs/en/hooks |
| **Claude Code — サブエージェント** | https://code.claude.com/docs/en/sub-agents |
| **Claude Code — MCP統合** | 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 — ヘッドレス（非インタラクティブモード）** | https://code.claude.com/docs/en/headless |
| **プロンプトエンジニアリングガイド** | https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/overview |
| **拡張思考** | https://platform.claude.com/docs/en/build-with-claude/extended-thinking |
| **Anthropic Cookbook（コードサンプル）** | https://github.com/anthropics/anthropic-cookbook |

---

# 第I部：理論的基礎

この部分では、試験に合格するために必要な理論をすべてカバーします。内容は試験のドメインではなく、技術と概念ごとに整理されています。これにより、各トピックについてより深い理解を構築できます。

---

# 第1章：Claude API — モデル操作の基礎

> ドキュメント：Messages API | プロンプトエンジニアリング

## 1.1 APIリクエストの構造

Claude APIはリクエスト-レスポンスモデルに従います。Claude Messages APIへの各リクエストには以下が含まれます。

```
{
  "model": "claude-sonnet-4-6",
  "max_tokens": 1024,
  "system": "あなたは親切なアシスタントです。",
  "messages": [
    {"role": "user", "content": "こんにちは！"},
    {"role": "assistant", "content": "こんにちは！"},
    {"role": "user", "content": "調子はどうですか？"}
  ],
  "tools": [...],
  "tool_choice": {"type": "auto"}
}
```

**主要フィールド：**

- `model` — モデル選択（`claude-opus-4-6`、`claude-sonnet-4-6`、`claude-haiku-4-5`）

- `max_tokens` — レスポンスの最大トークン数
- `system` — システムプロンプト（モデルの動作を定義）
- `messages` — 会話履歴（**一貫性を保つために毎回履歴全体を送信する必要があります**）
- `tools` — 利用可能なツールの定義
- `tool_choice` — ツール選択戦略

## 1.2 メッセージのロール

`messages`配列は2つの会話用ロールと1つの指示用ロールを使用します。

- `user` — ユーザーメッセージ。ツール結果を含みます（`user`ロールのメッセージ内の`tool_result`コンテンツブロックとして送信され、独立した`tool`ロールとしては送信されません）
- `assistant` — モデルのレスポンス（履歴送信時に含める）。ツール使用リクエスト（`tool_use`コンテンツブロック）を含みます
- `system` — トップレベルの`system`フィールドで設定する（最初のターンから適用）か、`messages`内に`{"role": "system", ...}`として挿入する（その時点以降に適用され、配置ルールに従います — 下記参照）ことができます

ツール結果は`"tool"`ロールのメッセージとして送信されるわけではありません。`user`ロールのメッセージとして送信され、その内容に`tool_result`コンテンツブロックが含まれます。

```
{
  "role": "user",
  "content": [
    {
      "type": "tool_result",
      "tool_use_id": "toolu_01...",
      "content": "..."
    }
  ]
}
```

`system`は、トップレベルの`system`パラメータだけでなく、`messages`配列内に直接ロールとして現れることもあります。これは、トップレベルの`system`フィールドのキャッシュされたプレフィックスを無効にすることなく、会話の途中で指示を追加するためのものです。これには特定の配置ルールがあります。

- `user`ターン（`tool_result`ブロックを含むものも含む）、またはサーバーツール使用で終わる`assistant`ターンの直後に置く必要があります。
- `assistant`ターンの前に置くか、配列の末尾に置く必要があります。
- `tool_use`ブロックとその`tool_result`の間に置くことはできません — そうすると400エラーが返されます。
- 後続の`system`メッセージ（会話途中のものを含む）は、それ以前のメッセージや、後続のターンに対するトップレベルの`system`フィールドよりも優先されます。

**重要：** すべてのAPIリクエストで**会話履歴全体**を送信する必要があります。モデルはリクエスト間で状態を保持しません—各呼び出しは独立しています。

## 1.3 レスポンスの`stop_reason`フィールド

Claude APIのレスポンスには`stop_reason`が含まれ、モデルが生成を停止した理由を示します。

| 値 | 説明 | 対応 |
|---|---|---|
| `"end_turn"` | モデルがレスポンスを完了した | 結果をユーザーに表示する |
| `"tool_use"` | モデルがツールを呼び出したい | ツールを実行して結果を返す |
| `"max_tokens"` | トークン制限に達した | レスポンスが切り捨てられています。制限の引き上げが必要な場合があります |
| `"stop_sequence"` | 停止シーケンスに遭遇した | アプリケーションロジックに基づいて処理する |

エージェントシステムでは、`"tool_use"`と`"end_turn"`が最も重要です—これらがエージェントループを制御します。

## 1.4 システムプロンプト

システムプロンプトは、コンテキストと動作ルールを定義する特殊な命令です。

- `messages`配列には含まれず、`system`フィールドに別途渡します
- ユーザーメッセージより優先されます
- 一度ロードされ、会話全体に適用されます
- ロール、制約、出力形式を定義するために使用します

**試験のポイント：** システムプロンプトの文言は、意図しないツールの関連付けを生む可能性があります。例えば「常に顧客を確認してください」という命令は、不要な場合でもモデルが`get_customer`を過剰に使用する原因になります。

## 1.5 コンテキストウィンドウ

コンテキストウィンドウは、モデルが一度に処理できるテキストの総量（トークン単位）です。以下が含まれます。

- システムプロンプト
- メッセージ履歴全体
- ツール定義
- ツール実行結果

**主なコンテキストウィンドウの問題：**

1. **中間消失効果（Lost-in-the-middle）：** モデルは長い入力の先頭と末尾の情報は確実に処理しますが、中間の詳細を見落とすことがあります。対策：重要な情報を先頭または末尾に配置します。

2. **ツール結果の蓄積：** ツール呼び出しのたびに出力がコンテキストに追加されます。ツールが40以上のフィールドを返しても5つしか必要でない場合、コンテキストの大部分が無駄になります。
3. **逐次要約：** 履歴を圧縮すると、数値、パーセンテージ、日付などが失われ、「約」「おおよそ」「いくつか」といった曖昧な表現になりがちです。

---

# 第2章：ツールと`tool_use`

> ドキュメント：Tool Use

## 2.1 `tool_use`とは

`tool_use`はClaudeが外部関数を呼び出せるようにするメカニズムです。モデルは直接コードを実行せず、構造化されたツール呼び出しリクエストを生成します。あなたのコードがそれを実行し、結果を返します。

## 2.2 ツール定義

各ツールはJSONスキーマを使って定義します。

```
{
  "name": "get_customer",
  "description": "メールアドレスまたはIDで顧客を検索します。名前、メール、注文履歴、アカウントステータスを含む顧客プロフィールを返します。注文確認前に顧客の身元を確認するため、lookup_orderの前に本ツールを使用してください。メールアドレス（形式：user@domain.com）または数値のcustomer_idを受け付けます。",
  "input_schema": {
    "type": "object",
    "properties": {
      "email": {"type": "string", "description": "顧客のメールアドレス"},
      "customer_id": {"type": "integer", "description": "数値の顧客ID"}
    },
    "required": []
  }
}
```

**ツールの説明における重要な点：**

1. **説明が主要な選択メカニズムです。** LLMはツールの説明に基づいてツールを選択します。最小限の説明（「顧客情報を取得します」）は、ツールが重複している場合にミスを招きます。
2. **説明に含めるべき内容：**
   - ツールが何をするか、何を返すか
   - 入力形式とサンプル値
   - エッジケースと制約

    - 類似する代替ツールではなくこのツールを使うタイミング
3. **避けるべきこと：** 複数のツールで同一または重複する説明を使用しないこと。`analyze_content`と`analyze_document`の説明がほぼ同じであれば、モデルは混乱します。
4. **組み込みツールとMCPツール：** エージェントは、類似する機能を持つMCPツールよりも組み込みツール（Read、Grep）を好む場合があります。これを防ぐには、MCPツールの説明を強化します—組み込みツールが提供できない具体的な利点、固有のデータ、またはコンテキストを強調します。

## 2.3 `tool_choice`パラメータ

`tool_choice`はモデルのツール選択方法を制御します。

| 値 | 動作 | 使用タイミング |
|---|---|---|
| `{"type": "auto"}` | モデルがツールを呼び出すかテキストで回答するかを決定 | ほとんどのケースのデフォルト |
| `{"type": "any"}` | モデルは**必ず**何らかのツールを呼び出す | 構造化出力の保証が必要な場合 |
| `{"type": "tool", "name": "extract_metadata"}` | モデルは**必ず**特定のツールを呼び出す | 強制的な最初のステップが必要な場合 |

**重要なシナリオ：**

- `tool_choice: "any"` + 複数の抽出ツール → モデルが最適なものを選択するが、構造化出力は保証される
- 強制選択 → 特定の最初のアクションを保証する必要がある場合（例：`extract_metadata`を強化前に実行）

## 2.4 構造化出力のためのJSONスキーマ

JSONスキーマを使った`tool_use`は、Claudeから構造化出力を得る**最も信頼性の高い**方法です。

- 構文的に有効なJSONを保証します（閉じ括弧の欠落、末尾のカンマなし）
- 必要な構造を強制します（必須フィールドが存在する）
- 意味的な正確性は**保証しません**（値は依然として間違っている可能性があります）

**スキーマ設計の主要原則：**

```
{
  "type": "object",
  "properties": {
    "category": {
      "type": "string",
      "enum": ["bug", "feature", "docs", "unclear", "other"]
    },
    "category_detail": {
      "type": ["string", "null"],
      "description": "category = 'other'または'unclear'の場合の詳細"
    },
    "severity": {
      "type": "string",
      "enum": ["critical", "high", "medium", "low"]
    },
    "confidence": {
      "type": "number",
      "minimum": 0,
      "maximum": 1
    },
    "optional_field": {
      "type": ["string", "null"],
      "description": "ソースで情報が見つからない場合はNull"
    }
  },
  "required": ["category", "severity"]
}
```

**スキーマ設計ルール：**

1. **必須 vs オプション：** 情報が常に利用可能な場合にのみフィールドを必須としてください。必須フィールドは、データが欠けている場合にモデルに値を捏造させます。
2. **Nullableフィールド：** 存在しない可能性がある情報には`"type": ["string", "null"]`を使用します。モデルは幻覚を起こす代わりに`null`を返せます。
3. **`"other"`を含むenum：** 事前定義したカテゴリ以外のデータを失わないよう、`"other"` + 詳細文字列を追加します。
4. **`"unclear"`のenum：** モデルが自信を持ってカテゴリを選択できない場合のために—誠実な`"unclear"`は間違ったカテゴリよりも優れています。

## 2.5 構文エラーと意味エラー

| エラータイプ | 例 | 軽減策 |
|---|---|---|
| **構文** | 無効なJSON、フィールドの型が間違っている | JSONスキーマを使った`tool_use`（排除できる） |
| **意味** | 合計が合わない、値が間違ったフィールドにある、幻覚 | 検証チェック、フィードバック付きの再試行、自己修正 |

---

# 第3章：Claude Agent SDK — エージェントシステムの構築

> ドキュメント：Agent SDK | Hooks | Subagents | Sessions

## 3.1 エージェントループとは

エージェントループは、自律的なタスク実行のための中核パターンです。モデルは単に回答するのではなく、一連のアクションを実行します。

```
1. ツールとともにClaudeにリクエストを送信
2. レスポンスを受信
3. stop_reasonを確認：
   - "tool_use" -> ツールを実行し、結果を履歴に追加し、ステップ1に戻る
   - "end_turn" -> タスク完了。結果をユーザーに表示
4. 完了するまで繰り返す
```

**これはモデル主導のアプローチです：** Claudeはコンテキストと先行するツール結果に基づいて、次に呼び出すツールを決定します。これはアクションの順序が固定されているハードコードされた決定木とは異なります。

**アンチパターン（避けるべきもの）：**

- 完了を検出するためにアシスタントのテキストを解析すること（「タスク完了」など）
- 主要な停止条件として任意の反復制限を使用すること（例：`max_iterations=5`）
- アシスタントがテキストコンテンツを生成したかどうかを完了シグナルとして確認すること

**正しいアプローチ：** 唯一の信頼性のある完了シグナルは`stop_reason == "end_turn"`です。

## 3.2 `AgentDefinition`の設定

`AgentDefinition`はClaude Agent SDKのエージェント設定オブジェクトです。

```
agent = AgentDefinition(
    name="customer_support",
    description="返品と注文問題の顧客リクエストを処理する",
    system_prompt="あなたはカスタマーサポートエージェントです...",
    allowed_tools=["get_customer", "lookup_order", "process_refund",
"escalate_to_human"],
    # コーディネーターの場合：
    # allowed_tools=["Task", "get_customer", ...]
)
```

**主要パラメータ：**

- `name` / `description` — エージェントの識別と説明

- `system_prompt` — 指示を含むシステムプロンプト
- `allowed_tools` — 許可されたツールのリスト（最小権限の原則）

## 3.3 ハブアンドスポーク：コーディネーターとサブエージェント

マルチエージェントアーキテクチャは通常、ハブアンドスポークトポロジーとして構築されます。

```
        コーディネーター
       /      |       \
  サブエージェント1  サブエージェント2  サブエージェント3
   （検索）     （分析）     （統合）
```

**コーディネーターの責務：**

- タスクをサブタスクに分解する
- 必要なサブエージェントを決定する（動的選択）
- サブエージェントに作業を委譲する
- 結果を集約・検証する
- エラーと再試行を処理する
- ユーザーに結果を伝える

**重要な原則：サブエージェントはコンテキストが分離されています。**

- サブエージェントはコーディネーターの会話履歴を自動的に引き継ぎません
- 必要なコンテキストはすべて**明示的に**サブエージェントのプロンプトに渡す必要があります
- サブエージェントは呼び出し間でメモリを共有しません
- すべての通信はコーディネーターを通じて行われます（可観測性とエラー制御のため）

## 3.4 サブエージェントを起動するための `Task` ツール

サブエージェントは`Task`ツールを介して起動されます。

```python
# コーディネーターのallowedToolsに"Task"を含める必要があります
coordinator_agent = AgentDefinition(
    allowed_tools=["Task", "get_customer"]
)
```

**明示的なコンテキスト受け渡しが必須です：**

```
# 悪い例：サブエージェントにコンテキストがない
Task: "ドキュメントを分析してください"

# 良い例：プロンプトにコンテキスト全体を含める
Task: "以下のドキュメントを分析してください。
ドキュメント: [ドキュメント全文]
先行する検索結果: [Web検索結果]
出力形式の要件: [スキーマ]"
```

**並列起動：** コーディネーターは1つのレスポンスで複数の`Task`を呼び出せます—サブエージェントは並列で実行されます。

```
# 1つのコーディネーターレスポンスに含まれる：
Task 1: "Xについての記事を検索してください"
Task 2: "ドキュメントYを分析してください"
Task 3: "Zについての記事を検索してください"
# 3つすべてが並列に実行される
```

## 3.5 Agent SDKのフック

フックはエージェントライフサイクルの特定のポイントで傍受・変換を可能にします。

**PostToolUse**はツール結果をモデルに提供する前に傍受します：

```python
# 例：異なるMCPツールからの日付形式を正規化する
@hook("PostToolUse")
def normalize_dates(tool_result):
    # Unixタイムスタンプ -> ISO 8601に変換
    # "Mar 5, 2025" -> "2025-03-05"に変換
    return normalized_result
```

**発信呼び出し傍受フック**はポリシーに違反するアクションをブロックします：

```python
# 例：500ドルを超える払い戻しをブロック
@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)
```

**主要な違い：フック vs プロンプト指示**

| 属性 | フック | プロンプト指示 |
|---|---|---|
| 保証 | **決定論的**（100%） | **確率論的**（90%以上、100%ではない） |
| 使用タイミング | 重要なビジネスルール、金融操作、コンプライアンス | 一般的な好み、推奨事項、書式設定 |
| 例 | 500ドルを超える払い戻しをブロック | 「エスカレーション前に解決を試みてください」 |

**ルール：** 失敗が財務的、法的、または安全上の結果をもたらす場合は、プロンプトではなくフックを使用してください。

# 第4章：Model Context Protocol（MCP）

> ドキュメント：MCP | Tools | Resources | Servers

## 4.1 MCPとは

Model Context Protocol（MCP）は、外部システムをClaudeに接続するためのオープンプロトコルです。MCPは3つの主要なリソースタイプを定義します。

1. **ツール** — エージェントがアクションを実行するために呼び出せる関数（CRUD操作、API呼び出し、コマンド実行）
2. **リソース** — エージェントがコンテキストのために読み取れるデータ（ドキュメント、データベーススキーマ、コンテンツカタログ）
3. **プロンプト** — 一般的なタスク向けの事前定義されたプロンプトテンプレート

## 4.2 MCPサーバー

MCPサーバーはMCPプロトコルを実装し、ツール/リソースを提供するプロセスです。MCPサーバーに接続すると：

- すべてのツールが自動的に検出されます
- 接続されたすべてのサーバーのツールが同時に使用可能になります
- ツールの説明がモデルの使用方法を決定します

## 4.3 MCPサーバーの設定

**プロジェクト設定（`.mcp.json`）** — チームでの使用向け：

```
{
  "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}"
      }
    }
  }
}
```

**主要なポイント：**

- `.mcp.json` はプロジェクトのルートに保存され、バージョン管理で管理されます
- 環境変数（`${GITHUB_TOKEN}`）はシークレット用に使用されます—トークン自体はコミットしません
- すべてのプロジェクト貢献者が利用できます

**ユーザー設定（`~/.claude.json`）** — 個人用/実験用サーバー：

- ユーザーのホームディレクトリに保存されます
- バージョン管理で共有されません
- 個人的な実験とテストに適しています

**サーバーの選択：**

- 標準的な統合（Jira、GitHub、Slack）には既存のコミュニティMCPサーバーを優先します
- ユニークでチーム固有のワークフローの場合にのみ独自サーバーを構築します

## 4.4 MCPの `isError` フラグ

MCPツールがエラーに遭遇した場合、レスポンスに `isError: true` を使用します。これはエージェントに呼び出しが失敗したことを通知します。

**構造化されたエラー（良い例）：**

```
{
  "isError": true,
  "content": {
    "errorCategory": "transient",
    "isRetryable": true,
    "message": "サービスは一時的に利用できません。注文APIへの呼び出しがタイムアウトしました。",
    "attempted_query": "order_id=12345",
    "partial_results": null
  }
}
```

**一般的なエラー（アンチパターン）：**

```
{
  "isError": true,
  "content": "操作が失敗しました"
}
```

一般的なエラーはエージェントに意思決定のための情報を与えません—再試行すべきか、クエリを変更すべきか、エスカレーションすべきか？

## 4.5 MCPリソース

リソースはエージェントがアクションを実行せずにコンテキストを取得するために要求できるデータです。

- コンテンツカタログ（例：全プロジェクトタスクのリスト、階層的なナビゲーション）
- データベーススキーマ（データ構造の理解）
- ドキュメント（APIリファレンス、内部ガイド）
- 課題/タスクのサマリー

**リソースの利点：** エージェントは何のデータが存在するかを把握するための探索的なツール呼び出しを必要としません。リソースは即座に「マップ」を提供します。

---

# 第5章：Claude Code — 設定とワークフロー

> ドキュメント：Claude Code | メモリ/CLAUDE.md | スキル | MCP | Hooks | サブエージェント | GitHub Actions | ヘッドレス

## 5.1 CLAUDE.mdの階層

CLAUDE.mdはClaude Codeの命令ファイルです。3レベルの階層があります。

```
1. ユーザーレベル：~/.claude/CLAUDE.md
    - そのユーザーにのみ適用
    - VCSで共有されない
    - 個人的な好みと作業スタイル

2. プロジェクトレベル：.claude/CLAUDE.md またはルートのCLAUDE.md
    - すべてのプロジェクト貢献者に適用
    - VCSで管理
    - コーディング標準、テスト標準、アーキテクチャの決定

3. ディレクトリレベル：サブディレクトリのCLAUDE.md
    - そのディレクトリのファイルを操作するときに適用
    - そのコードベース部分に特有の規約
```

**よくあるミス：** 新しいチームメンバーがプロジェクト指示を受け取れない場合、それは`.claude/CLAUDE.md`（プロジェクトレベル）ではなく`~/.claude/CLAUDE.md`（ユーザーレベル）に配置されているためです。

## 5.2 `@path`構文（ファイルインポート）

CLAUDE.mdは`@path`を使って外部ファイルを参照でき、設定をモジュール化できます。

```
# プロジェクトのCLAUDE.md

コーディング標準は @./standards/coding-style.md に記載されています
テスト要件は @./standards/testing-requirements.md にあります
プロジェクト概要は @README.md で依存関係は @package.json にあります
```

**`@path`のルール：**

- ファイルパスの直前に`@`を使用します（スペースなし）
- 相対パスと絶対パスの両方がサポートされています
- 相対パスはインポートを含むファイルからの相対パスとして解決されます
- 最大インポートネスト深度は5です

これにより重複を避け、各パッケージに関連する標準のみを含めることができます。

## 5.3 `.claude/rules/`ディレクトリ

`.claude/rules/`はモノリシックなCLAUDE.mdの代替で、トピックごとにルールを整理するために使用します。

```
.claude/rules/
  testing.md          -- テスト規約
  api-conventions.md  -- API規約
  deployment.md       -- デプロイルール
  react-patterns.md   -- Reactパターン
```

**主要機能：条件付きロードのための`paths`を含むYAMLフロントマター：**

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

APIファイルには、明示的なエラーハンドリングを含むasync/awaitを使用してください。
各エンドポイントは標準的なレスポンスラッパーを返す必要があります。
```

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

テストはdescribe/itブロックを使用する必要があります。
ハードコーディングではなくデータファクトリーを使用してください。
データベースをモックしないでください–テスト用データベースを使用してください。
```

**動作原理：**

- ルールはClaude Codeが`paths`パターンにマッチするファイルを編集する場合に**のみ**ロードされます
- これによりコンテキストとトークンを節約できます—関連のないルールはロードされません
- Globパターンにより、場所に関係なくファイルタイプで規約を適用できます（コードベース全体に散在するテストに最適）

**`paths`を使った`.claude/rules/`とディレクトリレベルのCLAUDE.mdの使い分け：**

- `.claude/rules/`と`paths` — 規約が多くのディレクトリに散在するファイルに適用される場合（テスト、マイグレーション）
- ディレクトリレベルのCLAUDE.md — 規約が特定のディレクトリに関連し、他では不要な場合

## 5.4 カスタムスラッシュコマンドとスキル

> **注意：** 現行のClaude Codeバージョンでは、カスタムコマンド（`.claude/commands/`）はスキル（`.claude/skills/`）と統合されています。どちらの形式も`/name`コマンドを作成します。試験ガイドは`.claude/commands/`を参照していますが、その形式は引き続きサポートされています。

スラッシュコマンドは`/name`で呼び出す再利用可能なプロンプトテンプレートです。

**`.claude/commands/`形式（レガシー、サポート継続中）：**

```
.claude/commands/
  review.md        -- /review -- 標準コードレビュー
  test-gen.md      -- /test-gen -- テスト生成
```

**`.claude/skills/`形式（現行）：**

```
.claude/skills/
  review/SKILL.md  -- /review -- フロントマター設定あり
  test-gen/SKILL.md
```

**プロジェクトコマンド**（`.claude/commands/`または`.claude/skills/`）：

- VCSに保存され、リポジトリをクローンしたときに全員が利用できます
- チーム全体で一貫したワークフローを確保します

**ユーザーコマンド**（`~/.claude/commands/`または`~/.claude/skills/`）：

- VCSで共有されない個人コマンド
- 個人ワークフロー用

## 5.5 スキル — `.claude/skills/`

スキルはSKILL.mdフロントマターで設定される高度なコマンドです。

```
---
context: fork
allowed-tools: ["Read", "Grep", "Glob"]
argument-hint: "分析するディレクトリへのパス"
---

指定されたディレクトリのコード構造を分析してください。
依存関係とアーキテクチャパターンに関するレポートを出力してください。
```

**フロントマターパラメータ：**

| パラメータ | 説明 |
|---|---|
| `context: fork` | 隔離されたサブエージェントでスキルを実行します。詳細な出力がメインセッションを汚染しません |
| `allowed-tools` | 利用可能なツールを制限します（セキュリティ—例：スキルが許可されていなければファイルを削除できません） |
| `argument-hint` | パラメータなしで呼び出された際に引数を求めるヒント |

**スキルとCLAUDE.mdの使い分け：**

- **スキル** — 特定のタスク（レビュー、分析、生成）のオンデマンド呼び出し
- **CLAUDE.md** — 常にロードされる一般的な標準と規約

**個人スキル（`~/.claude/skills/`）：**

- チームメンバーに影響しないよう、別の名前で個人的なバリアントを作成します

## 5.6 計画モード vs 直接実行

**計画モード：**

- モデルは調査と計画のみを行います。変更は加えません
- Read、Grep、Globを使ってコードベースを探索します
- ユーザーが承認する実装計画を作成します
- 副作用のない安全な探索

**計画モードを使うタイミング：**

- 大規模な変更（数十のファイル）
- 複数の実行可能なアプローチがある場合（マイクロサービス：どこでサービス境界を引くか？）
- アーキテクチャの決定（どのフレームワーク？どの構造？）
- 慣れないコードベース（変更前に理解する必要がある）
- 45ファイル以上に影響するライブラリマイグレーション

**直接実行を使うタイミング：**

- 明確なスタックトレースを持つ単一ファイルの修正
- 1つの検証チェックの追加
- よく理解された、明確な変更

**組み合わせアプローチ：**

1. 調査と設計のための計画モード
2. ユーザーが計画を承認
3. 承認された計画を実装するための直接実行

**探索サブエージェント** — コードベース探索のための専門化されたサブエージェント：

- 詳細な出力をメインコンテキストから隔離します
- 概要のみを返します
- マルチフェーズタスクでのコンテキストウィンドウの枯渇を防ぎます

## 5.7 `/compact` コマンド

`/compact` はコンテキストを圧縮するための組み込みコマンドです。

- 以前の履歴を要約してコンテキストウィンドウを解放します
- 詳細なツール出力でコンテキストが満たされた長い調査セッションで使用します
- リスク：要約中に正確な数値、日付、特定の詳細が失われる可能性があります

## 5.8 `/memory` コマンド

`/memory` はセッション間でメモリを管理するための組み込みコマンドです。

- `CLAUDE.md` ファイルを編集用に開き、メモ、好み、コンテキストを保存できます
- 情報はセッション間で保持され、起動時に自動的にロードされます

- プロジェクト規約、ユーザーの好み、よく使うコマンド、現在の作業コンテキストの保存に便利です
- 毎回のセッションで同じ指示を再説明する代わりの手段です

## 5.9 CI/CD向けClaude Code CLI

**`-p`（または`--print`）フラグ：**

```
claude -p "このプルリクエストのセキュリティ問題を分析してください"
```

- 非インタラクティブモード：プロンプトを処理し、stdoutに出力し、終了します
- ユーザー入力を待ちません
- CI/CDパイプラインでClaudeを実行する唯一の正しい方法です

**CI向けの構造化出力：**

```
claude -p "このPRをレビューしてください" --output-format json --json-schema
'{"type":"object",...}'
```

- `--output-format json` — JSON形式で出力
- `--json-schema` — スキーマに対して出力を検証
- 結果を解析してPRインラインコメントを自動投稿できます

**セッションコンテキストの分離：** コードを生成したClaudeセッションは、それをレビューする際の効果が低い傾向があります（モデルは自分の推論コンテキストを保持しており、自分の決定に異議を唱えにくくなっています）。レビューには独立したインスタンスを使用してください。

**重複コメントの防止：** 新しいコミット後に再レビューする際は、先行するレビュー結果をコンテキストに含め、新規または未解決の問題のみを報告するよう指示してください。

## 5.10 `fork_session`とセッション管理

**`--resume <session-name>`**は名前付きセッションを再開します：

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

- 保存されたコンテキストを持つ前の会話を継続します
- 複数のセッションにまたがる長期的な調査に便利です
- リスク：前のセッション以降にファイルが変更された場合、ツール結果が古くなっている可能性があります

**`fork_session`**は共有コンテキストから独立したブランチを作成します：

```
コードベース調査
         |
    fork_session
    /           \
アプローチA:      アプローチB:
Redux            Context API
```

- 両方のフォークは分岐点までのコンテキストを引き継ぎます
- 以降は独立して分岐します
- アプローチの比較や戦略のテストに便利です

**再開ではなく新しいセッションを開始するタイミング：**

- ツール結果が古い（ファイルが変更された）
- 長い時間が経過してコンテキストが劣化した
- 古いツールデータで再開するよりも「以下が判明したことの概要です：...」で再開する方が良い場合

---

# 第6章：プロンプトエンジニアリング — 高度なテクニック

> ドキュメント：プロンプトエンジニアリング | Anthropic Cookbook

## 6.1 Few-shotプロンプティング

Few-shotプロンプティングは、期待される動作を示すために2〜4つの入力/出力サンプルをプロンプトに含めることです。

**Few-shotがテキスト説明より効果的な理由：**

- 「もっと正確に」といった曖昧な指示は様々な解釈が可能です
- サンプルは期待される形式と決定ロジックを明確に示します
- モデルは新しいケースにパターンを汎化します（サンプルを単に繰り返すわけではない）

**Few-shotサンプルの種類と使用タイミング：**

1. **曖昧なシナリオ向けのサンプル：**

```
リクエスト：「注文が壊れています」
アクション：get_customer -> lookup_order -> ステータス確認を呼び出す
理由：「壊れている」は破損した商品を意味する可能性があります。注文の詳細が必要です。

リクエスト：「マネージャーを呼んでください」
アクション：escalate_to_humanを即座に呼び出す
理由：顧客が明示的に人間を求めています。自律的な解決を試みないでください。
```

2. **出力形式向けのサンプル：**

```
発見のサンプル：
{
  "location": "src/auth/login.ts:42",
  "issue": "usernameパラメータのSQLインジェクション",
  "severity": "critical",
  "suggested_fix": "パラメータ化クエリを使用する"
}
```

3. **許容可能なコードと問題のあるコードを区別するサンプル：**

```
// 許容可能（フラグを立てない）：
const items = data.filter(x => x.active);

// 問題あり（フラグを立てる）：
const items = data.filter(x => x.active == true); // 厳密等価 === を使用すること
```

4. **異なるドキュメント形式からの抽出サンプル：**

```
インライン引用を含むドキュメント：
「研究によると（Smith、2023）、率は42%です。」
-> {"value": "42%", "source": "Smith, 2023", "type": "inline_citation"}

参考文献付きのドキュメント：
「率は42%です。[1]」
-> {"value": "42%", "source": "reference_1", "type": "bibliography"}
```

5. **非公式な測定値のサンプル：**

```
テキスト：「ご飯をひとつかみ程度」
-> {"amount": "~100g", "original_text": "ひとつかみ", "precision": "approximate"}

テキスト：「塩ひとつまみ」
-> {"amount": "~1g", "original_text": "ひとつまみ", "precision": "approximate"}
```

Few-shotは、純粋なルールベースの指示では対応しきれないほど多様な、非公式で非標準的な測定単位の抽出に特に効果的です。

**プロンプトの形式正規化ルール：** 構造化出力に厳格なJSONスキーマを使用する場合は、プロンプトに正規化ルールを追加します。

```
正規化：
- 日付：常にISO 8601（YYYY-MM-DD）。「昨日」-> 絶対日付を計算する
- 通貨：数値金額 + 通貨コード。「5ドル」-> {"amount": 5, "currency": "USD"}
- パーセンテージ：十進分数。「半分」-> 0.5
```

これにより、JSONは構文的に有効でも値が一貫していないという意味エラーを防ぎます。

## 6.2 明示的な基準 vs 曖昧な指示

**悪い例（曖昧）：**

```
コードコメントの正確性を確認してください。
保守的に—高信頼度の発見のみを報告してください。
```

**良い例（明示的な基準）：**

```
コメントが問題ありとフラグを立てるのは以下の場合のみ：
1. コメントが実際のコードの動作と矛盾する動作を説明している場合
2. コメントが存在しない関数や変数を参照している場合
3. TODO/FIXMEコメントがコードで既に修正されたバグを指している場合

フラグを立てないこと：
- 単にスタイル的に古くなっているコメント
- 軽微な表現の不正確さを含むコメント
- 欠落しているコメント（それは別のカテゴリ）
```

**コード例を含む深刻度基準を定義する：**

```
CRITICAL：ユーザーのランタイム障害
  例：支払い処理中のNullPointerException

HIGH：セキュリティの脆弱性
  例：SQLインジェクション、XSS、認可チェックの欠如

MEDIUM：即時影響のないロジックバグ
  例：間違ったソート、オフバイワンエラー

LOW：コード品質
  例：重複、小規模データに対する非最適アルゴリズム
```

## 6.3 プロンプトチェーニング

プロンプトチェーニングは複雑なタスクを集中した一連のステップに分割します。

```
ステップ1：auth.tsを分析（ローカルの問題のみ）
        -> 出力：auth.tsの問題リスト

ステップ2：database.tsを分析（ローカルの問題のみ）
        -> 出力：database.tsの問題リスト

ステップ3：統合パス（クロスファイルの依存関係）
        -> 出力：モジュール境界の問題
```

**なぜこれが重要か：**

- **注意力の希薄化**を防ぎます—モデルが一度に多くのファイルを受け取ると、一部のファイルのバグを見逃しながら他のファイルに浅いコメントを提供する可能性があります
- ファイルごとに一貫した分析品質を確保します
- クロスファイルの相互作用の分析を別途可能にします

**プロンプトチェーニング vs 動的分解の使い分け：**

- **プロンプトチェーニング** — 予測可能で繰り返し可能なタスク（コードレビュー、ファイルマイグレーション）
- **動的分解** — 実行中にサブタスクが明らかになるオープンエンドの調査

## 6.4 「インタビュー」パターン

ソリューションを実装する前に、Claudeが明確化の質問をします。

```
Claude：「APIのキャッシングを実装する前に、いくつか質問があります：
1. キャッシュ無効化戦略はTTLとイベントベースどちらを好みますか？
2. キャッシュが利用できない場合、古いデータは許容されますか？
3. キャッシングはユーザーごとですか、グローバルですか？
4. キャッシュするデータの予想量はどのくらいですか？」
```

**これが有用なタイミング：**

- 馴染みのないドメイン（フィンテック、医療、法律システム）
- 明白でない含意を持つタスク（キャッシュ戦略、障害モード）
- 文脈によって最善の選択が変わる複数の実行可能アプローチ

## 6.5 検証と再試行（フィードバック付き）

抽出されたデータが検証に失敗した場合：

```
ステップ1：ドキュメントからデータを抽出する
ステップ2：検証（Pydantic、JSONスキーマ、ビジネスルール）
ステップ3：エラーがある場合–コンテキストとともに再試行：
  - 元のドキュメント
  - 前の（誤った）抽出結果
  - 具体的なエラー：「フィールド'total' = 150ですが、sum(line_items) = 145です。値を再確認してください。」
```

**再試行が効果的なタイミング：**

- 形式エラー（間違った形式の日付）
- 構造エラー（フィールドが間違った場所に置かれている）
- 計算的な不一致（モデルは再確認できる）

**再試行が効果的でないタイミング：**

- 情報がソースドキュメントに存在しない
- 必要なコンテキストが外部にある（データが提供されていない別のドキュメントにある）

**Pydanticをバリデーションツールとして：** Pydanticは、スキーマベースのデータ検証のためのPythonライブラリです。試験のポイント：

- **構造的バリデーション：** ClaudeからJSONを受け取った後、コードで型、必須性、enum制約を確認します
- **意味的バリデーション：** カスタムバリデーターがビジネスロジックを強制します（商品の合計が総計と一致する、start_date < end_dateなど）

- **検証-再試行ループ：** Pydanticの検証失敗時に、エラーコンテキストとともにClaudeに再プロンプトします
- **JSONスキーマ生成：** Pydanticモデルは`tool_use`用のJSONスキーマを生成でき、単一のソースオブトゥルースを提供します

## 6.6 自己修正

内部の矛盾を検出するパターン：

```
{
  "stated_total": "$150.00",
  "calculated_total": "$145.00",
  "conflict_detected": true,
  "line_items": [
    {"name": "ウィジェットA", "price": 75.00},
    {"name": "ウィジェットB", "price": 70.00}
  ]
}
```

モデルは記載された値と計算された値の両方を抽出します—差異がある場合、`conflict_detected`によって不一致を処理できます。

---

# 第7章：Message Batches API

> ドキュメント：Message Batches

## 7.1 概要

Message Batches APIは非同期処理のためのリクエストバッチを送信できます。

| 属性 | 値 |
|---|---|
| コスト削減 | 同期呼び出し比 **50%** |
| 処理ウィンドウ | 最大 **24時間**（レイテンシSLAの保証なし） |
| マルチターンツール呼び出し | **サポートされていない**（1リクエスト = 1レスポンス） |
| 相関付け | リクエストとレスポンスをリンクするための`custom_id`フィールド |

## 7.2 Batch API vs 同期APIの使い分け

| タスク | API | 理由 |
|---|---|---|

| | | |
|---|---|---|
| マージ前のPRチェック | **同期** | 開発者が待っています。24時間は許容できません |
| 夜間の技術的負債レポート | **バッチ** | 朝までに結果が必要。50%の節約 |
| 週次のセキュリティ監査 | **バッチ** | 緊急ではない。50%の節約 |
| インタラクティブなコードレビュー | **同期** | 即時レスポンスが必要 |
| 10,000件のドキュメントの処理 | **バッチ** | 大量処理。節約が大きい |

## 7.3 `custom_id`の使用

```
{
  "custom_id": "doc-invoice-2024-001",
  "params": {
    "model": "claude-sonnet-4-6",
    "max_tokens": 1024,
    "messages": [{"role": "user", "content": "以下からデータを抽出してください：..."}]
  }
}
```

`custom_id`によって以下が可能になります：

- 結果を元のドキュメントにリンクする
- 失敗した場合、失敗したドキュメントのみを再送信する
- 成功したドキュメントの再処理を避ける

## 7.4 バッチの失敗処理

1. 100件のドキュメントのバッチを送信
2. 95件が成功、5件が失敗（コンテキスト制限超過）
3. `custom_id`で失敗を特定
4. 戦略を変更する（例：長いドキュメントをチャンクに分割）
5. 5件の失敗したドキュメントのみを再送信

## 7.5 SLA計画

30時間以内に結果が必要で、Batch APIに最大24時間かかる可能性がある場合：

- 送信ウィンドウ：30 - 24 = **6時間**
- バッチはデッドラインの24時間以内に送信する必要があります
- 頻繁な送信の場合、4時間のウィンドウに分割します

---

# 第8章：タスク分解戦略

## 8.1 固定パイプライン（プロンプトチェーニング）

各ステップが事前に定義されています：

```
ドキュメント -> メタデータ抽出 -> データ抽出 -> 検証 -> エンリッチメント -> 最終出力
```

**使用タイミング：**

- タスク構造が予測可能（レビューは常に同じテンプレートに従う）
- すべてのステップが事前に判明している
- 安定性と再現性が必要

## 8.2 動的適応的分解

サブタスクは中間結果に基づいて生成されます：

```
1. 「レガシーコードベースにテストを追加する」
2. -> まず：構造をマッピングする（Glob、Grep）
3. -> 発見：テストなしの3モジュール、部分的なカバレッジの2モジュール
4. -> 優先付け：支払いモジュールから始める（高リスク）
5. -> 作業中：外部APIへの依存関係を発見
6. -> 適応：テストを書く前に外部APIのモックを追加する
```

**使用タイミング：**

- オープンエンドの調査タスク
- 全体的な範囲が事前に不明な場合
- 各ステップが前のステップの結果に依存する場合

## 8.3 マルチパスのコードレビュー

10以上のファイルを含むプルリクエストの場合：

```
パス1（ファイルごと）：auth.tsを分析 -> ローカルの問題リスト
パス1（ファイルごと）：database.tsを分析 -> ローカルの問題リスト
パス1（ファイルごと）：routes.tsを分析 -> ローカルの問題リスト
...
パス2（統合）：ファイル間の関係を分析
  -> クロスファイルの問題：一貫性のない型、循環依存
```

**14ファイルを1パスで見るのが悪い理由：**

- 注意力の希薄化：一部のファイルは深い分析、他は浅い分析
- 一貫性のないコメント：1つのファイルではパターンにフラグを立てるが、別のファイルでは承認する
- バグの見落とし：明白なエラーが認知的過負荷でスキップされる

# 第9章：エスカレーションと人間のループへの組み込み

## 9.1 人間へのエスカレーションのタイミング

**エスカレーショントリガー（明確なルール）：**

| 状況 | アクション |
|---|---|
| 顧客が「マネージャーを呼んでください」と明示的に要求する | 即座にエスカレーション。解決を試みない |
| ポリシーがリクエストをカバーしていない | エスカレーション（例：ポリシーが競合他社の価格マッチングについて沈黙している場合） |
| エージェントが進展できない | 妥当な試行回数後にエスカレーション |
| 閾値を超える金融操作 | エスカレーション（プロンプトではなくフックで強制するのが好ましい） |
| 顧客検索で複数一致 | 追加の識別子を求める。推測しない |

**信頼できないトリガー：**

| 信頼できない方法 | 失敗する理由 |
|---|---|
| 感情分析 | 顧客の気分はケースの複雑さと相関しない |
| モデルの自己評価信頼度（1〜10） | モデルは自信を持って間違える可能性がある。キャリブレーションが不十分 |
| 自動分類器 | 過剰なエンジニアリング。訓練データが必要な場合がある |

## 9.2 エスカレーションパターン

**即時エスカレーション：**

```
顧客：「マネージャーと話したい」
エージェント：[即座にescalate_to_humanを呼び出す]
NG：「問題についてお手伝いできます、少々お待ちください...」
```

**解決試行後のエスカレーション：**

```
顧客：「購入2日後に冷蔵庫が壊れました」
エージェント：[注文を確認し、保証交換を提案]
顧客が不満な場合 -> エスカレーション
```

**段階的なエスカレーション（認識 → 解決 → 再度要求でエスカレーション）：**

```
顧客：「これはひどい。品質に非常に不満です！」
エージェント：[不満を認識]「ご不満をお察しします。」
        [解決策を提案]「交換または返金をご提案できます。」
顧客：「いいえ、誰かと話したい！」
エージェント：[顧客が再度主張する  -> 即座にエスカレーション]
```

主要原則：まず感情を認識し、次に具体的な解決策を提案し、顧客が人間を求めて繰り返す場合にのみエスカレーションします。不満の最初の表明でエスカレーションしません（それはマネージャーを求めることとは同じではありません）。

**ポリシーギャップでのエスカレーション：**

```
顧客：「競合他社Xがこのアイテムを30%安く販売しています—割引をください」
ポリシー：自社サイトでの価格調整のみカバー
エージェント：[エスカレーション — ポリシーは競合他社の価格マッチングをカバーしていません]
```

## 9.3 構造化された引き継ぎプロトコル

エスカレーション時、エージェントは人間に構造化されたサマリーを渡すべきです：

```
{
  "customer_id": "CUST-12345",
  "customer_name": "田中太郎",
  "issue_summary": "破損商品の返金リクエスト",
  "order_id": "ORD-67890",
  "root_cause": "商品が破損した状態で到着。写真添付",
  "actions_taken": [
    "get_customerで顧客を確認",
    "lookup_orderで注文を確認",
    "標準交換を提案 — 顧客は返金を主張"
  ],
  "refund_amount": "¥12,980",
  "recommended_action": "全額返金を承認",
  "escalation_reason": "顧客がマネージャーとの対話を要求"
}
```

人間のオペレーターは会話の全文記録にアクセスできません—このサマリーのみが見えます。そのため、完全で自己完結型である必要があります。

## 9.4 信頼度キャリブレーションと人間の監視

データ抽出システムの場合：

1. **フィールドレベルの信頼度スコア：** モデルは抽出されたフィールドごとに信頼度スコアを出力します
2. **キャリブレーション：** ラベル付きの検証セットを使ってしきい値を調整します
3. **ルーティング：**
   - 高信頼度 + 安定した精度 -> 自動処理
   - 低信頼度または曖昧なソース -> 人間によるレビュー

**層別ランダムサンプリング：**

- 高信頼度の抽出でも定期的にサンプルを監査する
- 全体97%の精度が特定のドキュメントタイプで40%のエラーを隠している可能性がある
- 全体だけでなく、ドキュメントタイプとフィールドごとの精度を分析する

---

# 第10章：マルチエージェントシステムのエラーハンドリング

## 10.1 エラーカテゴリ

| カテゴリ | 例 | 再試行可能 | エージェントのアクション |
|---|---|---|---|
| **一時的** | タイムアウト、503、ネットワーク障害 | はい | 指数バックオフで再試行 |
| **バリデーション** | 無効な入力形式、必須フィールド欠如 | いいえ（入力を修正） | リクエストを変更して再試行 |
| **ビジネス** | ポリシー違反、閾値超過 | いいえ | ユーザーに説明し、代替案を提案 |
| **権限** | アクセス拒否 | いいえ | エスカレーション |

## 10.2 エラーハンドリングのアンチパターン

| アンチパターン | 問題 | 正しいアプローチ |
|---|---|---|
| 汎用ステータス「検索利用不可」 | コーディネーターが回復方法を決定できない | エラータイプ、クエリ、部分的な結果、代替案を返す |
| サイレント抑制（空の結果 = 成功） | コーディネーターは一致がなかったと思うが、実際は失敗だった | 「結果なし」を「検索失敗」と区別する |
| 1つの失敗でワークフロー全体を中断 | すべての部分的な結果を失う | 部分的な結果で継続し、ギャップを注釈する |
| サブエージェント内での無限再試行 | レイテンシとリソースの無駄 | ローカルリカバリ（1〜2回の再試行）、その後コーディネーターに伝播 |

## 10.3 構造化されたサブエージェントエラー

```
{
  "status": "partial_failure",
  "failure_type": "timeout",
  "attempted_query": "2024年のAIによる音楽業界への影響",
  "partial_results": [
    {"title": "AIによる音楽生成レポート", "url": "...", "relevance": 0.8}
  ],
  "alternative_approaches": [
    "より絞り込んだクエリを試す：'AI音楽作曲ツール'",
    "代替データソースを使用する"
  ],
  "coverage_impact": "未カバー：AI音楽制作への影響"
}
```

これによりコーディネーターは以下を決定するための情報を得られます：

- 変更されたクエリで再試行するか？
- 部分的な結果を使用するか？
- 別のサブエージェントに委譲するか？
- このセクションなしで続行し、ギャップを注釈するか？

## 10.4 最終統合でのカバレッジ注釈

```
## レポート：創造的産業へのAIの影響

### 視覚芸術（完全カバレッジ）
[調査結果]

### 音楽（部分カバレッジ — 検索エージェントのタイムアウト）
[部分的な結果]
⚠️ 注意：検索エージェントのタイムアウトのため、このセクションのカバレッジは限定的です。

### 文学（完全カバレッジ）
[調査結果]
```

---

# 第11章：本番システムのコンテキスト管理

## 11.1 事実を別のブロックに抽出する

会話履歴（要約時に劣化する）に依存する代わりに、重要な事実を構造化されたブロックに抽出します。

```
=== ケースの事実（新しい事実が現れるたびに更新） ===
顧客ID：CUST-12345
注文ID：ORD-67890
注文日：2025-01-15
注文金額：¥12,980
問題：配送時の商品破損
顧客の要求：全額返金
ステータス：マネージャーの承認待ち
===
```

履歴がどのように要約されても、このブロックをすべてのプロンプトに含めます。

## 11.2 ツール結果のトリミング

`lookup_order`が40以上のフィールドを返しても現在のタスクに必要なのは5つだけの場合：

```python
# PostToolUseフック：関連フィールドのみを保持
@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"]
    }
```

これによりコンテキストを節約し、ノイズを軽減します。

## 11.3 位置を意識した入力

中間消失効果を念頭に置いて重要な情報を配置します：

```
[主要な発見 – 先頭]
3つの重大な脆弱性が見つかりました...

[詳細な結果 – 中間]
=== auth.ts ファイル ===
...
=== database.ts ファイル ===
...

[アクション項目 – 末尾]
優先事項：マージ前にauth.tsの脆弱性を修正。
```

## 11.4 スクラッチパッドファイル

長い調査では、エージェントは重要な発見をスクラッチパッドファイルに書き込めます：

```
# investigation-scratchpad.md
## 主要な発見
- src/payments/processor.tsのPaymentProcessorはBaseProcessorを継承
- refund()は3か所から呼び出される：OrderController、AdminPanel、CronJob
- 外部PaymentGateway APIのレート制限：100リクエスト/分
- マイグレーション#47でrefund_reason（NOT NULL）を追加 — 2024-12-01
```

コンテキストが劣化した場合（または新しいセッションで）、エージェントは再発見を実行する代わりにスクラッチパッドを参照できます。

## 11.5 コンテキストを保護するためのサブエージェントへの委譲

```
メインエージェント：「支払いモジュールの依存関係を調査してください」
  -> サブエージェント（探索）：15ファイルを読み込み、インポートを追跡
  -> 返す：「支払いはAuthService、OrderModel、外部PaymentGateway APIに依存しています」

メインエージェント：15ファイルの代わりに1行をコンテキストに保持
```

**別のコンテキスト層：** マルチエージェントシステムでは、各サブエージェントは限られたコンテキストバジェット内で動作します—タスクに必要な情報のみを受け取ります。コーディネーターは別のコンテキスト層として機能します：サブエージェントの出力を集約し、グローバル状態を保存し、コンテキストを割り当てます。これにより、あるエージェントが他のエージェントと無関係な情報でウィンドウを消費する「コンテキストの漏洩」を防ぎます。

**サブエージェントの制約付きコンテキストバジェット：**

- 最小限のコンテキストを送信：特定のタスク + 必要なデータ
- サブエージェントに構造化された結果を返すよう指示する（生データのダンプではなく）
- `allowedTools`を使ってサブエージェントのツールセットを制限する—ツールが少ないほど気が散らず、コストも低い

## 11.6 構造化された状態の永続化（クラッシュリカバリのため）

各エージェントは自分の状態を既知の場所にエクスポートします：

```json
// agent-state/web-search-agent.json
{
  "status": "completed",
  "queries_executed": ["AI音楽 2024", "AI音楽作曲"],
  "results_count": 12,
  "key_findings": [...],
  "coverage": ["音楽作曲", "音楽制作"],
  "gaps": ["音楽配信", "音楽ライセンス"]
}
```

コーディネーターは再開時にマニフェストをロードします：

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

---

# 第12章：出所の保持

## 12.1 帰属の喪失問題

複数のソースの結果を要約する場合、「主張 → ソース」のリンクが失われる可能性があります：

```
悪い例：「AI音楽市場は32億ドルと推定されています。」（ソースなし、年次なし）

良い例：
{
  "claim": "AI音楽市場は32億ドルと推定されています。",
  "source_url": "https://example.com/report",
  "source_name": "グローバルAI音楽レポート2024",
  "publication_date": "2024-06-15",
  "confidence": 0.9
}
```

## 12.2 矛盾するデータの処理

2つのソースが異なる値を提供する場合：

```
{
  "claim": "ストリーミングプラットフォームにおけるAI生成音楽のシェア",
  "values": [
    {
      "value": "12%",
      "source": "Spotifyアニュアルレポート2024",
      "date": "2024-03",
      "methodology": "自動分類"
    },
    {
      "value": "8%",
      "source": "音楽業界協会調査",
      "date": "2024-07",
      "methodology": "500レーベルの調査"
    }
  ],
  "conflict_detected": true,
  "possible_explanation": "方法論と期間の違い"
}
```

任意に1つの値を選択しないでください。両方を帰属とともに保持し、コーディネーターに決定させます。

## 12.3 正確な解釈のための日付の記載

日付がないと、時間的な差異が矛盾として誤解される可能性があります：

```
悪い例：「ソースAは10%、ソースBは15%と言っています。矛盾。」
良い例：「ソースA（2023）は10%、ソースB（2024）は15%と言っています。1年間で5%の成長が考えられます。」
```

## 12.4 コンテンツタイプ別のレンダリング

すべてを1つの形式に強制しないでください：

- 財務データ -> テーブル
- ニュースと分析 -> 散文
- 技術的な発見 -> 構造化されたリスト
- 時系列 -> 時系列順

---

# 第13章：Claude Codeの組み込みツール

## 13.1 ツール選択リファレンス

| タスク | ツール | 例 |
| --- | --- | --- |

| | | |
|---|---|---|
| 名前/パターンでファイルを検索 | **Glob** | `**/*.test.tsx`、`src/components/**/*.ts` |
| ファイル内を検索 | **Grep** | 関数名、エラーメッセージ、インポート |
| ファイルを全文読み込む | **Read** | 分析のためにファイルをロード |
| 新しいファイルを書き込む | **Write** | ゼロからファイルを作成 |
| 既存ファイルを正確に編集 | **Edit** | 一意のテキスト一致で特定のスニペットを置換 |
| シェルコマンドを実行 | **Bash** | git、npm、テスト実行、ビルド |

## 13.2 漸進的な調査戦略

一度にすべてのファイルを読み込まないでください。漸進的に理解を構築します：

```
1. Grep：エントリポイントを見つける（関数定義、エクスポート）
2. Read：見つかったファイルを読む
3. Grep：使用箇所を見つける（インポート、呼び出し）
4. Read：コンシューマーファイルを読む
5. 完全な全体像が得られるまで繰り返す
```

## 13.3 フォールバック：EditのかわりのRead + Write

一意のテキスト一致がないためEditが失敗した場合：

1. Read — ファイルの全内容をロード
2. プログラム的にコンテンツを変更
3. Write — 更新されたバージョンを書き込む

---

# 第II部：試験ドメインノート

---

# ドメイン1：エージェントアーキテクチャとオーケストレーション（27%）

## 1.1 自律的なタスク実行のためのエージェントループの設計

### 主要知識：

- エージェントループのライフサイクル：`stop_reason`（`"tool_use"` vs `"end_turn"`）を確認し、ツールを実行し、次のイテレーションに向けて結果を返す
- ツール結果は会話履歴に追加され、モデルが次のアクションを決定できる

- モデル主導の意思決定（Claudeが次のツールを選択）vs ハードコードされた決定木

### 主要スキル：

- フロー制御：`stop_reason = "tool_use"`でループを継続し、`"end_turn"`で停止する
- イテレーション間でツール結果をコンテキストに追加する
- 避けるべきアンチパターン：完了を検出するためのアシスタントテキストの解析、主要な停止メカニズムとして任意の反復制限を使用すること

## 1.2 マルチエージェントシステム（コーディネーター-サブエージェント）のオーケストレーション

### 主要知識：

- ハブアンドスポークアーキテクチャ：コーディネーターがすべてのエージェント間通信、エラーハンドリング、ルーティングを所有する
- サブエージェントはコンテキストが分離されている—コーディネーターの履歴を自動的に引き継がない
- コーディネーターの責務：タスク分解、委譲、結果集約、サブエージェントの動的選択
- コーディネーターによる過度に狭い分解のリスク

### 主要スキル：

- 重複を最小化するためにサブエージェント間で調査カバレッジを分割する
- 反復的な改善ループを実装する（コーディネーターが統合を評価し、タスクを再ルーティングする）
- すべての通信を可観測性のためにコーディネーターを通じてルーティングする

## 1.3 サブエージェント呼び出し、コンテキスト受け渡し、起動の設定

### 主要知識：

- `Task`ツールがサブエージェントを起動する。コーディネーターの`allowedTools`に`"Task"`を含める必要がある
- サブエージェントコンテキストはプロンプトに明示的に含める必要がある。サブエージェントは親のコンテキストを引き継がない
- `AgentDefinition`の設定：説明、システムプロンプト、ツールの制約
- 代替案を探索するための`fork_session`によるセッション管理

### 主要スキル：

- 先行するエージェントの出力全体をサブエージェントプロンプトに含める
- コンテキストを渡す際にデータとメタデータを区別するための構造化形式を使用する
- 1つのコーディネーターターンで複数の`Task`呼び出しを通じて並列サブエージェントを起動する
- ステップバイステップの指示ではなく、目標と品質基準の観点でコーディネーターのプロンプトを書く

## 1.4 強制と引き継ぎパターンを使ったマルチステップワークフローの実装

### 主要知識：

- **プログラム的な強制**（フック、前提条件）と**プロンプトガイダンス**の違い
- 決定論的な保証が必要な場合（例：金融操作前の身元確認）、プロンプトだけでは不十分
- エスカレーション時の構造化された引き継ぎプロトコル（顧客ID、理由、推奨アクション）

### 主要スキル：

- 先行するステップが完了するまで後続の呼び出しをブロックするプログラム的な前提条件（例：`get_customer`が確認済みIDを返すまで`process_refund`をブロック）
- 複数の側面を持つ顧客リクエストを別々のアイテムに分解する
- 人間にエスカレーションする際に構造化されたサマリーを生成する

## 1.5 ツール呼び出しの傍受とデータ正規化のためのAgent SDKフック

### 主要知識：

- モデルが消費する前にツール結果を傍受するフックパターン（例：`PostToolUse`）
- コンプライアンスルールを強制するために発信呼び出しを傍受するフック（例：閾値を超える払い戻しをブロック）
- フックは**決定論的な保証**を提供するが、プロンプト指示は**確率論的なコンプライアンス**を提供する

### 主要スキル：

- データ形式を正規化するための`PostToolUse`フック（Unixタイムスタンプ、ISO 8601、数値ステータスコード）
- エスカレーションへのリダイレクトでポリシー違反アクションをブロックする傍受フック
- ビジネスルールが保証されたコンプライアンスを必要とする場合は、プロンプトよりフックを選択する

## 1.6 複雑なワークフローのためのタスク分解戦略

### 主要知識：

- **固定パイプライン**（プロンプトチェーニング）vs 中間結果に基づく**動的適応的分解**
- プロンプトチェーニング：逐次ステップ（各ファイルを別々に分析し、その後統合パスを実行）
- 発見されたものに基づいてサブタスクを生成する適応的な調査計画

### 主要スキル：

- 予測可能なマルチ側面レビューにはプロンプトチェーニング、オープンエンドの調査には動的分解を使用する
- 大規模なコードレビューをファイルごとの分析と別のクロスファイル統合パスに分割する

- オープンエンドのタスクを分解する：まず構造をマッピングし、次に優先順位付けされた計画を構築する

## 1.7 セッション状態、再開、フォーキング

### 主要知識：

- `--resume <session-name>`で名前付きセッションを継続する
- 共有コンテキストから独立した調査ブランチを作成するための`fork_session`
- セッションを再開する際のエージェントへのファイル変更の通知の重要性
- 古い結果での再開よりも構造化されたサマリーを使った新しいセッションの方が信頼性が高い場合がある

### 主要スキル：

- 名前付き調査セッションを継続するために`--resume`を使用する
- 代替アプローチを並列比較するために`fork_session`を使用する
- 再開（コンテキストが現在有効）と新しいセッション開始（結果が古い）の選択

---

# ドメイン2：ツール設計とMCP統合（18%）

## 2.1 明確な説明を持つツールインターフェースの設計

### 主要知識：

- ツールの説明は**LLMがツールを選択するための主要メカニズム**です。最小限の説明は信頼性のない選択につながります
- 入力形式、サンプルクエリ、エッジケース、適用境界を含めることの重要性
- 曖昧または重複する説明は誤ルーティングを引き起こす
- システムプロンプトの文言は意図しないツールの関連付けを生む可能性がある

### 主要スキル：

- 各ツールを類似する代替案から明確に区別する説明を書く
- 機能の重複を排除するためにツール名を変更する（例：`analyze_content` -> `extract_web_results`）
- 汎用ツールを明確な入力/出力コントラクトを持つ特化されたツールに分割する

## 2.2 MCPツールの構造化されたエラーレスポンスの実装

### 主要知識：

- MCPツールのレスポンスにある`isError`フラグ

- **一時的なエラー**（タイムアウト）、**バリデーションエラー**（不正な入力）、**ビジネスエラー**（ポリシー違反）、**アクセス/権限エラー**の違い
- 汎用エラー（「操作が失敗しました」）は正しい回復の決定を妨げる
- 再試行可能なエラーと再試行不可能なエラーの違い

### 主要スキル：

- `errorCategory`（一時的/バリデーション/権限）、`isRetryable`、人間が読めるメッセージなどの構造化メタデータを返す
- ビジネスルール違反には`retryable: false`と明確なユーザー向け説明を使用する
- 一時的な障害のサブエージェント内でのローカルリカバリ。解決できないエラーのみを伝播する
- アクセス失敗（再試行の決定が必要）と有効な空の結果（一致なし）を区別する

## 2.3 エージェント間のツール割り当てと`tool_choice`の設定

### 主要知識：

- エージェントごとのツールが多すぎる（4〜5つの代わりに18個など）と、ツール選択の信頼性が**低下**する
- 専門外のツールを持つエージェントはそれらを誤用する傾向がある
- スコープ付きのツールアクセス：ロール関連のツールのみと、限られたクロスロールユーティリティ
- `tool_choice`：`"auto"`、`"any"`、強制ツール選択（`{"type": "tool", "name": "..."}`）

### 主要スキル：

- 各サブエージェントのツールセットをそのロールに関連するものに制限する
- 汎用ツールを制約されたものに置き換える（例：`fetch_url` -> `load_document`）
- テキスト回答の代わりにツール呼び出しを保証するために`tool_choice: "any"`を使用する
- 実行順序を確保するために特定のツールを強制する

## 2.4 Claude CodeとエージェントワークフローへのMCPサーバーの統合

### 主要知識：

- MCPサーバーのスコープ：チーム向けのプロジェクト（`.mcp.json`）vs 実験向けのユーザー（`~/.claude.json`）
- シークレット管理のための`.mcp.json`での環境変数の置換（例：`${GITHUB_TOKEN}`）
- 接続されたすべてのMCPサーバーのツールは接続時に自動検出され、同時に利用可能になる
- 探索的なツール呼び出しを減らすための「コンテンツカタログ」（タスクサマリー、データベーススキーマ）としてのMCPリソース

### 主要スキル：

- 環境変数ベースのトークンを使ってプロジェクト`.mcp.json`に共有MCPサーバーを設定する
- 個人/実験的なサーバーは`~/.claude.json`に保持する

- 標準的な統合にはカスタムサーバーよりコミュニティMCPサーバーを優先する

## 2.5 組み込みツール（Read、Write、Edit、Bash、Grep、Glob）の選択と適用

### 主要知識：

- **Grep**：ファイルの内容を検索する（関数名、エラーメッセージ、インポート）
- **Glob**：名前/拡張子パターンでファイルを見つける
- **Read/Write**：フルファイル操作。**Edit**：一意のテキスト一致による正確な変更
- 一意のマッチがないためEditが失敗した場合は、Read + Writeにフォールバック

### 主要スキル：

- コンテンツ検索にはGrepを、パターンによるファイル検出にはGlobを使用する
- 漸進的に理解を構築する：エントリポイントをGrepで見つけ、フローを追跡するためにReadする
- ラッパーモジュールを通じて関数の使用を追跡する

---

# ドメイン3：Claude Codeの設定とワークフロー（20%）

## 3.1 階層、スコープ、モジュール組織によるCLAUDE.mdの設定

### 主要知識：

- CLAUDE.mdの階層：ユーザー（`~/.claude/CLAUDE.md`）、プロジェクト（`.claude/CLAUDE.md`またはルートの`CLAUDE.md`）、ディレクトリレベル（サブディレクトリのCLAUDE.md）
- ユーザーレベルの設定は1人のユーザーにのみ適用され、VCSで共有されない
- CLAUDE.mdをモジュール化するための外部ファイルの参照に使う`@path`構文（例：`@./standards/coding-style.md`）
- モノリシックなCLAUDE.mdの代わりにトピック重点のルールファイルを持つ`.claude/rules/`ディレクトリ

### 主要スキル：

- 階層の問題を診断する（新しいチームメンバーがプロジェクトレベルではなくユーザーレベルにあるため指示を受け取れない）
- 各パッケージのCLAUDE.mdに標準を選択的に含めるために`@path`（例：`@./standards/testing.md`）を使用する
- 大きなCLAUDE.mdを複数の`.claude/rules/`ファイルに分割する（testing.md、api-conventions.md、deployment.md）

## 3.2 カスタムスラッシュコマンドとスキルの作成と設定

### 主要知識：

- **プロジェクトコマンド**は`.claude/commands/`（VCSで共有）vs **ユーザーコマンド**は`~/.claude/commands/`
- `SKILL.md`フロントマターを使った`.claude/skills/`のスキル：`context: fork`、`allowed-tools`、`argument-hint`
- `context: fork`はメインセッションを汚染しないよう、隔離されたサブエージェントコンテキストでスキルを実行する
- 個人のスキルバリアントは`~/.claude/skills/`に別の名前で置ける

### 主要スキル：

- チーム全員が受け取れるようにプロジェクトのスラッシュコマンドを`.claude/commands/`に保存する
- 詳細な出力を持つスキルを隔離するために`context: fork`を使用する
- スキルが使用できるツールを制限するために`allowed-tools`を使用する
- 必須パラメータを開発者に求めるために`argument-hint`を使用する

## 3.3 条件付き規約ロードのためのパス固有のルールの使用

### 主要知識：

- `.claude/rules/`ファイルはYAMLフロントマターの`paths`を含め、globパターンに基づいてルールを有効化できる
- パス固有のルールは一致するファイルを編集する場合に**のみ**ロードされ、コンテキストとトークンを節約する
- 規約がコードベース全体に散在する多くのディレクトリのファイルに適用される場合（例：テスト）、ディレクトリレベルのCLAUDE.mdよりGlobベースのパスルールが好ましい

### 主要スキル：

- 一致するファイルを操作するときのみロードされるよう`paths: ["terraform/**/*"]`を含む`.claude/rules/`ファイルを作成する
- ディレクトリに関係なくファイルタイプで規約を適用するためにGlobパターン（`**/*.test.tsx`）を使用する
- 規約がコードベース全体にまたがる場合は、ディレクトリレベルのCLAUDE.mdよりパス固有のルールを優先する

## 3.4 計画モードと直接実行の使い分け

### 主要知識：

- **計画モード**：大規模な変更、複数の実行可能なアプローチ、アーキテクチャの決定を伴う複雑なタスク向け
- **直接実行**：シンプルでよく理解された変更向け（例：単一のバリデーションの追加）

- 計画モードは変更前のコードベースの安全な探索を可能にする
- 探索サブエージェントは詳細な発見出力を隔離する

### 主要スキル：

- アーキテクチャ的な影響を持つタスクには計画モードを使用する（マイクロサービス、45以上のファイルに触れるマイグレーション）
- 明確なスタックトレースと単一ファイルの修正には直接実行を使用する
- マルチフェーズタスクでのコンテキストウィンドウ枯渇を防ぐために探索サブエージェントを使用する
- アプローチを組み合わせる：発見のための計画モード、実装のための実行モード

## 3.5 段階的な改善のための反復的な洗練

### 主要知識：

- 具体的な入力/出力サンプルは期待を伝える最も効果的な方法
- **テスト駆動の反復**：まずテストを書き、失敗に基づいて反復する
- 「インタビュー」パターン：Claudeが質問をして明確でない設計上の考慮事項を表面化する
- 1つのメッセージですべての問題を提供する場合（相互依存）vs 順次提供する場合（独立）

### 主要スキル：

- 変換要件を明確にするために2〜3の具体的な入力/出力サンプルを提供する
- 実装前に期待される動作、エッジケース、パフォーマンス要件のテストセットを構築する
- 設計の側面（キャッシュ無効化、障害モード）を表面化するためにインタビューパターンを使用する
- エッジケースのためのサンプル入力と期待される出力を含む具体的なテストケースを提供する

## 3.6 CI/CDパイプラインへのClaude Codeの統合

### 主要知識：

- 自動化されたパイプラインの非インタラクティブモード向けの`-p`（または`--print`）フラグ
- CI向けの構造化出力のための`--output-format json`と`--json-schema`
- CLAUDE.mdはCIがトリガーするClaude Codeのプロジェクトコンテキスト（テスト標準、レビュー基準）を提供する
- **セッションコンテキストの分離**：コードを生成したセッションは、独立したインスタンスよりもレビューの効果が低い

### 主要スキル：

- インタラクティブな入力でのハングを避けるために`-p`を使ってCIでClaude Codeを実行する
- 構造化された結果（例：インラインPRコメント）のために`--output-format json` + `--json-schema`を使用する
- 新しいコミット後に再実行する際は、先行するレビュー結果を含める（新規または未修正の問題のみを報告）

- テスト生成の品質を向上させるためにCLAUDE.mdにテスト標準と利用可能なフィクスチャを文書化する
- 重複を避け、スタイルの一貫性を保つために新しいテストを生成する際にコンテキストに既存のテストファイルを含める

---

# ドメイン4：プロンプトエンジニアリングと構造化出力（20%）

## 4.1 精度向上のための明示的な基準を持つプロンプトの設計

### 主要知識：

- 明示的な基準は曖昧な指示より効果的（例：「コメントがコードと矛盾する場合のみフラグを立てる」vs「コメントの精度を確認する」）
- 「より保守的に」といった汎用的なガイダンスは、具体的なカテゴリ基準より効果が低い
- 一部カテゴリでの高い偽陽性率が開発者の信頼に与える影響：一部カテゴリでの高い偽陽性率は正確なカテゴリへの信頼も損なう

### 主要スキル：

- レビュー基準を定義する：何を報告するか（バグ、セキュリティ）vs 何を無視するか（軽微なスタイル）
- 高い偽陽性率のカテゴリを一時的に無効化する
- 各レベルのコード例を含む明示的な深刻度基準を定義する

## 4.2 出力の一貫性向上のためのFew-shotプロンプティングの使用

### 主要知識：

- Few-shotサンプルは一貫してフォーマットされた実用的な出力を生成する最も効果的な方法
- Few-shotは曖昧なケースの処理を示すことができる（ツール選択、テストカバレッジのギャップ）
- Few-shotはモデルが単にデフォルトを繰り返すのではなく新しいパターンに汎化するのを助ける
- Few-shotは抽出タスクでの幻覚を減らすことができる

### 主要スキル：

- 曖昧なシナリオに根拠を含む2〜4の的を絞ったサンプルを提供する
- 出力形式（場所、問題、深刻度、修正案）を示すFew-shotサンプルを含める
- 許容可能なコードパターンと実際の問題を区別するサンプルを提供する
- 異なる構造のドキュメントからの正確な抽出のサンプルを提供する

## 4.3 `tool_use`とJSONスキーマによる構造化出力の強制

### 主要知識：

- JSONスキーマを使った`tool_use`はスキーマに準拠した出力を保証し、JSON構文エラーを排除する最も信頼性の高い方法
- `tool_choice: "auto"`ではモデルはテキストを返す可能性がある。`"any"`ではツールを呼び出す必要がある。強制選択は特定のツールを選択する
- 厳格なJSONスキーマは構文エラーを排除するが、意味エラーは防がない（合計が合わない、値が間違ったフィールドにある）
- スキーマ設計：必須 vs オプションフィールド。拡張性のための「other」を含むenumと詳細文字列

### 主要スキル：

- JSONスキーマを持つ抽出ツールを定義し、`tool_use`結果からデータを解析する
- 複数のスキーマが存在する場合に構造化出力を保証するために`tool_choice: "any"`を使用する
- 特定のツール呼び出しを強制する：`tool_choice: {"type": "tool", "name": "extract_metadata"}`
- ソースに情報が含まれていない可能性がある場合に値の捏造を避けるためにフィールドをオプション/Nullableにする
- 拡張可能なカテゴリ化のために`"unclear"`や`"other"`のenum値と詳細フィールドを使用する

## 4.4 抽出品質のための検証、再試行、フィードバックループの実装

### 主要知識：

- フィードバック付き再試行：修正を導くために再試行プロンプトに具体的なバリデーションエラーを含める
- ソースに情報が存在しない場合、再試行は効果的でない
- フィードバックループの設計：発見をトリガーしたパターンを追跡する（`detected_pattern`）
- 意味エラー（合計が合わない）vs 構文エラー（`tool_use`で対処）

### 主要スキル：

- 元のドキュメント、誤った抽出、特定のバリデーションエラーを含むフォローアッププロンプト
- 再試行が効果的でない場合を特定する（必要な情報が別のドキュメントにしかない）
- 偽陽性を分析するために発見に`detected_pattern`フィールドを含める
- 不一致を検出するために`calculated_total`と`stated_total`の両方を抽出することによる自己修正の設計

## 4.5 効率的なバッチ処理戦略の設計

### 主要知識：

- Message Batches API：50%の節約、最大24時間の処理ウィンドウ、レイテンシSLAの保証なし

- バッチ処理はブロッキングでないタスク（夜間のレポート、監査）に適しており、ブロッキングタスク（マージ前チェック）には適していない
- Batch APIは単一リクエスト内でのマルチターンツール呼び出しをサポートしない
- バッチ内のリクエスト/レスポンスを相関付けるための`custom_id`フィールド

### 主要スキル：

- ブロッキングチェックには同期API、夜間/週次のワークロードにはBatch APIを使用する
- SLAの要件に基づいてバッチ送信のスケジュールを計画する（例：24時間処理の30時間保証のための4時間ウィンドウ）
- `custom_id`で特定された失敗したドキュメントのみを再送信することで失敗を処理する
- 大規模処理を実行する前にサンプルを使ってプロンプトを反復する

## 4.6 マルチインスタンスとマルチパスのレビューアーキテクチャの設計

### 主要知識：

- 自己レビューの限界：モデルは推論コンテキストを保持しており、自分の決定に異議を唱えにくい
- 独立したレビューインスタンス（生成コンテキストなし）は微妙な問題を見つけるのに優れている
- マルチパスレビュー：注意力の希薄化を避けるためのファイルごとのローカル分析と別のクロスファイル統合パス

### 主要スキル：

- 生成コンテキストなしで変更をレビューするために2番目の独立したClaudeインスタンスを使用する
- マルチファイルレビューをファイルごとのパスとクロスファイルのデータフロー分析のための統合パスに分割する
- キャリブレーションされた方法でレビューをルーティングするために自己評価信頼度を持つ検証パスを使用する

---

# ドメイン5：コンテキスト管理と信頼性（15%）

## 5.1 重要な情報を保持するための会話コンテキストの管理

### 主要知識：

- 逐次要約のリスク：数値、パーセンテージ、日付が曖昧なサマリーに圧縮される
- 中間消失効果：モデルは長い入力の先頭と末尾を確実に処理するが、中間の発見を見落とす可能性がある
- ツール出力は関連性に不釣り合いなコンテキストを蓄積できる（5つのみ必要なのに40以上のフィールド）
- 後続のAPIリクエストで完全な会話履歴を送信することの重要性

### 主要スキル：

- 要約された履歴の外の永続的な「ケースの事実」ブロックにトランザクションの事実を抽出する
- 詳細なツール出力を関連フィールドにトリミングする
- 明示的なセクション見出しを使って集約されたデータの先頭に主要な発見を配置する
- サブエージェントに構造化出力にメタデータ（日付、ソース）を含めるよう要求する

## 5.2 効果的なエスカレーションパターンの設計と曖昧さの解決

### 主要知識：

- 適切なエスカレーショントリガー：人間の明示的な要求、ポリシーのギャップ/例外、進展できない状況
- 即時エスカレーション（明示的な要求）vs 解決試行（エージェントのスコープ内）
- 感情分析とモデルの信頼度自己評価はケースの複雑さの信頼できないプロキシ
- 複数の顧客一致は推測的な推測ではなく追加の識別子を要求する必要がある

### 主要スキル：

- システムプロンプトにFew-shotサンプルを含む明示的なエスカレーション基準
- 追加調査なしに人間の明示的な要求を即座に実行する
- ポリシーが特定のリクエストに対して曖昧または沈黙している場合にエスカレーションする
- ツール結果に複数の一致が含まれる場合に追加の識別子を求める

## 5.3 マルチエージェントシステムのエラー伝播戦略の実装

### 主要知識：

- 構造化されたエラーコンテキスト（失敗タイプ、クエリ、部分的な結果、代替案）はよりスマートなコーディネーターのリカバリを可能にする
- アクセス失敗（タイムアウトは再試行の決定が必要）と有効な空の結果（一致なし）を区別する
- 汎用のエラーステータス（「検索利用不可」）はコーディネーターから価値あるコンテキストを隠す
- サイレント抑制または単一の失敗でワークフロー全体を中断するのはどちらもアンチパターン

### 主要スキル：

- 構造化されたエラーコンテキストを返す：失敗タイプ、試みたこと、部分的な結果、可能な代替案
- アクセス失敗と有効な空の結果を区別する
- 一時的な失敗のサブエージェント内でのローカルリカバリ。部分的な結果と共に回復不可能なエラーのみを伝播する
- 統合でのカバレッジを注釈する：十分にサポートされているものとギャップが残っているところ

## 5.4 大規模なコードベース調査時の効率的なコンテキスト管理

### 主要知識：

- 長いセッションでのコンテキスト劣化：モデルが不安定な回答を出し始め、特定のクラスではなく「典型的なパターン」を参照する
- スクラッチパッドファイルはコンテキスト境界を越えて重要な発見を保持する
- サブエージェントへの委譲は詳細な発見出力を隔離する
- 構造化された状態の永続化はクラッシュリカバリを可能にする

### 主要スキル：

- 特定の質問のためにサブエージェントを起動し、高レベルのコーディネーションをメインエージェントに保持する
- 重要な発見を保存し、後で参照するためにスクラッチパッドファイルを使用する
- 次のフェーズのサブエージェントを起動する前に重要な発見を要約する
- 長い調査中のコンテキスト使用量を減らすために`/compact`を使用する

## 5.5 人間の監視と信頼度キャリブレーションを持つワークフローの設計

### 主要知識：

- 集計メトリクス（例：全体97%の精度）は特定のドキュメントタイプやフィールドでの低いパフォーマンスを隠す可能性がある
- 層別ランダムサンプリングは高信頼度の抽出でのエラー率を測定する
- ラベル付きの検証セットを使ったフィールドレベルの信頼度キャリブレーション
- 自動化する前にドキュメントタイプとフィールドセグメントで精度を検証する

### 主要スキル：

- 新しいエラーパターンを検出するための層別ランダムサンプリングを実装する
- 安定したパフォーマンスを検証するためにドキュメントタイプとフィールドで精度を分析する
- フィールドレベルの信頼度スコアを出力し、ラベル付きデータを使ってレビューのしきい値を調整する
- 低信頼度または曖昧なソースの抽出を人間のレビューにルーティングする

## 5.6 マルチソース統合での出所の保持と不確実性の処理

### 主要知識：

- 「主張 → ソース」マッピングを保持せずに要約すると帰属が失われる
- 集約中に構造化されたマッピングを保持する必要がある
- 矛盾する統計は、任意に1つの値を選択するのではなく、帰属とともに矛盾を注釈することで処理する
- 時間的な差異を矛盾として誤解しないよう、公開/収集日付を含める

**主要スキル：**

- サブエージェントに「主張 → ソース」マッピング（URL、ドキュメント名、引用）を出力するよう要求する
- 確定的な発見と議論中の発見を分けるようにレポートを構造化する
- 矛盾する値を注釈とともに保持し、調整のためにコーディネーターに渡す
- 正確な時間的解釈のために公開日を含める
- タイプ別にコンテンツをレンダリングする：財務データはテーブル、ニュースは散文、技術的な発見は構造化されたリスト

---

# 試験問題のサンプルと解説

## 問題1（シナリオ：カスタマーサポートエージェント）

**状況：** データによると、12%のケースでエージェントが`get_customer`をスキップし、顧客の名前だけを使って`lookup_order`を呼び出し、誤った返金につながっています。

**最も効果的な変更は何ですか？**

- A) `get_customer`からIDを取得するまで`lookup_order`と`process_refund`をブロックするプログラム的な前提条件を追加する **【正解】**
- B) システムプロンプトを改善する
- C) Few-shotサンプルを追加する
- D) ルーティング分類器を実装する

**Aの理由：** 重要なビジネスロジックが特定のツール順序を必要とする場合、ソフトウェアはプロンプトベースのアプローチ（B、C）が提供できない**決定論的な保証**を提供します。Dはツールの順序ではなく可用性を扱っています。

---

## 問題2（シナリオ：カスタマーサポートエージェント）

**状況：** エージェントが注文関連の質問に対してよく`lookup_order`の代わりに`get_customer`を呼び出しています。ツールの説明は最小限で類似しています。

**最初のステップは何ですか？**

- A) Few-shotサンプル
- B) 各ツールの説明を入力形式、サンプル、境界で拡張する **【正解】**
- C) ルーティング層を追加する
- D) ツールをマージする

**Bの理由：** ツールの説明はモデルの主要な選択メカニズムです。これは最小労力で最大のインパクトをもたらす修正です。Aは根本原因に対処せずにトークンを追加します。Cは過剰なエンジニアリングです。Dは正当化されるよりも多くの努力を必要とします。

---

## 問題3（シナリオ：カスタマーサポートエージェント）

**状況：** エージェントが目標の80%に対してわずか55%の問題を解決しています。シンプルなケースをエスカレーションし、複雑なポリシーの例外を自律的に処理しようとします。

**キャリブレーションを改善する方法は？**

- A) Few-shotサンプルを含む明示的なエスカレーション基準を追加する **【正解】**
- B) 自動エスカレーションで自己評価信頼度（1〜10）
- C) 歴史的なデータで訓練された別の分類器
- D) 感情分析

**Aの理由：** 根本原因—シンプルと複雑なケース間の不明確な決定境界—に直接対処します。Bは信頼性がありません（モデルは自信を持って間違える可能性があります）。Cは過剰なエンジニアリングです。Dは別の問題（気分 ≠ 複雑さ）を解決します。

---

## 問題4（シナリオ：Claude Codeによるコード生成）

**状況：** リポジトリをクローンした際にチーム全体が利用できる標準コードレビュー用のカスタム`/review`コマンドが必要です。

**コマンドファイルはどこに作成すべきですか？**

- A) プロジェクトリポジトリの`.claude/commands/` **【正解】**
- B) `~/.claude/commands/`
- C) ルートの`CLAUDE.md`
- D) `.claude/config.json`

**Aの理由：** `.claude/commands/`に保存されたプロジェクトコマンドはバージョン管理され、全員が自動的に利用できます。Bは個人用コマンド用です。Cは指示のためのものであり、コマンド定義用ではありません。Dは存在しません。

---

## 問題5（シナリオ：Claude Codeによるコード生成）

**状況：** モノリスをマイクロサービスに再構築する必要があります（数十のファイル、サービス境界の決定）。

**どのアプローチを使うべきですか？**

- A) 計画モード：コードベースを探索し、依存関係を理解し、アプローチを設計する **【正解】**
- B) 漸進的に直接実行
- C) 詳細な事前指示で直接実行
- D) 困難になったら計画に切り替えながら直接実行

**Aの理由：** 計画モードは大規模な変更、複数の可能なアプローチ、アーキテクチャの決定のために設計されています。Bは高コストな手戻りのリスクがあります。Cはすでに構造を知っていることを前提としています。Dはリアクティブです。

## 問題6（シナリオ：Claude Codeによるコード生成）

**状況：** コードベースには異なる規約を持つ領域があります（React、API、データベース）。テストはコードと共に配置されています。規約を自動的に適用させたいです。

**どのアプローチを使うべきですか？**

- A) YAMLフロントマターとglobパターンを使った`.claude/rules/`ファイル **【正解】**
- B) すべてをルートのCLAUDE.mdに入れる
- C) `.claude/skills/`のスキル
- D) すべてのディレクトリにCLAUDE.md

**Aの理由：** globパターン（例：`**/*.test.tsx`）を使った`.claude/rules/`は、ファイルパスに基づいた自動的な規約適用を可能にします—コードベース全体に散在するテストに最適です。Bはモデルによる Inferenceに依存しています。Cは手動/オンデマンドです。Dは関連ファイルが多くのディレクトリにある場合にうまく機能しません。

---

## 問題7（シナリオ：マルチエージェント調査システム）

**状況：** システムが「クリエイティブ産業へのAIの影響」を調査しているが、レポートは視覚芸術のみをカバーしています。コーディネーターはトピックを「デジタルアートにおけるAI」「グラフィックデザインにおけるAI」「写真撮影におけるAI」に分解しました。

**原因は何ですか？**

- A) 統合エージェントがギャップを検出しない
- B) コーディネーターがタスクを過度に狭く分解した **【正解】**
- C) Web検索エージェントが十分に検索していない
- D) ドキュメント分析エージェントが非視覚的なソースをフィルタリングしている

**Bの理由：** ログはコーディネーターが「クリエイティブ産業」を視覚的なサブトピックのみに分解し、音楽、文学、映画を完全に見逃したことを示しています。サブエージェントは正しく実行されました—問題は何が割り当てられたかです。

---

## 問題8（シナリオ：マルチエージェント調査システム）

**状況：** Web検索サブエージェントが複雑なトピックの調査中にタイムアウトします。コーディネーターへのエラー情報の渡し方を設計する必要があります。

**どのエラー伝播アプローチがインテリジェントなリカバリを最もよく可能にしますか？**

- A) コーディネーターに構造化されたエラーコンテキストを返す：失敗タイプ、実行されたクエリ、部分的な結果、代替アプローチ **【正解】**
- B) サブエージェント内で指数バックオフで自動的に再試行し、再試行を使い果たした後に汎用の「検索利用不可」ステータスを返す

- C) サブエージェント内でタイムアウトをキャッチし、空の結果セットを成功としてマークして返す
- D) タイムアウト例外をトップレベルのハンドラーに伝播し、ワークフロー全体を終了する

**Aの理由：** 構造化されたエラーコンテキストはコーディネーターに、変更されたクエリで再試行するか、代替アプローチを試みるか、部分的な結果で続行するかを決定するために必要なものを提供します。Bは汎用ステータスの後ろにコンテキストを隠します。Cは失敗を成功としてマスクします。Dはワークフロー全体を不必要に中断します。

---

## 問題9（シナリオ：マルチエージェント調査システム）

**状況：** 統合エージェントは結果をマージしながら特定の主張を確認する必要があることが多い。コーディネーターを通じたやり取りにより、タスクごとに2〜3回の追加ラウンドトリップが発生し、レイテンシが40%増加します。評価によると、確認の85%は単純なファクトチェック（日付、名前、統計）で、15%はより深い調査が必要です。

**オーバーヘッドを削減しながら信頼性を維持するにはどうすればよいですか？**

- A) 単純なチェック向けに限定スコープの`verify_fact`ツールを統合エージェントに与え、複雑な確認はコーディネーターを通じてルーティングし続ける **【正解】**
- B) すべての確認ニーズを蓄積して最後にコーディネーターに一括返却する
- C) すべてのWeb検索ツールへの完全なアクセスを統合エージェントに与える
- D) 各ソースの周りに追加のコンテキストをプロアクティブにキャッシュする

**Aの理由：** これは最小権限の原則を適用します：統合エージェントは85%の一般的なケース（単純なファクトチェック）に正確に必要なものを得ながら、複雑な調査にはコーディネーター経由のパスを維持します。Bはブロッキング依存関係を導入します（後の統合ステップは以前に確認された事実に依存する可能性がある）。Cは責任の分離を壊します。Dは需要を確実に予測できない投機的キャッシュに依存しています。

---

## 問題10（シナリオ：Claude Code for CI）

**状況：** パイプラインが`claude "このプルリクエストのセキュリティ問題を分析してください"`を実行しているが、インタラクティブな入力を待ってハングします。

**正しいアプローチは何ですか？**

- A) `-p`フラグを使用する：`claude -p "このプルリクエストのセキュリティ問題を分析してください"` **【正解】**
- B) `CLAUDE_HEADLESS=true`を設定する
- C) `/dev/null`からstdinをリダイレクトする
- D) `--batch`を使用する

**Aの理由：** `-p`（または`--print`）はClaude Codeを非インタラクティブモードで実行するための文書化された方法です。プロンプトを処理し、stdoutに出力し、終了します。他のオプションは存在しない機能またはUnixの回避策です。

---

## 問題11（シナリオ：Claude Code for CI）

**状況：** チームはコスト削減のためにMessage Batches APIへの移行を検討しています。現在、2つのワークフローがあります：(1) PRマージ前に完了する必要があるブロッキングプリマージチェック、(2) 朝のレビュー用に夜間に生成される技術的負債レポート。マネージャーは両方をBatch APIに移行して50%を節約することを提案しています。

**この提案をどのように評価すべきですか？**

- A) 技術的負債レポートのみバッチ処理を使用し、プリマージチェックはリアルタイム呼び出しを維持する **【正解】**
- B) 両方のワークフローをバッチ処理に移行し、完了をポーリングする
- C) バッチ結果の順序付けの問題を避けるために両方のリアルタイム呼び出しを維持する
- D) バッチに時間がかかりすぎる場合のリアルタイムへのフォールバックで両方をバッチ処理に移行する

**Aの理由：** Message Batches APIは50%を節約しますが、処理時間は最大24時間かかる可能性があり、レイテンシSLAの保証がありません。これは開発者が待っているブロッキングプリマージチェックには不適切ですが、技術的負債レポートのような夜間のバッチワークロードには理想的です。

---

## 問題12（シナリオ：マルチファイルコードレビュー）

**状況：** プルリクエストが在庫追跡モジュールの14ファイルを変更しています。すべてのファイルを1回のパスでレビューすると、一貫性のない結果が生じます：一部のファイルには詳細なコメントがありますが、他のファイルには浅いコメントがあり、明らかなバグが見落とされ、矛盾したフィードバックがあります（1つのファイルでパターンに問題があるとフラグが立てられるが、同じコードが別のファイルでは承認される）。

**レビューをどのように再構築すべきですか？**

- A) 各ファイルをローカルの問題について個別に分析し、クロスファイルのデータフローのための別の統合パスを実行する **【正解】**
- B) 開発者に大きなPRを3〜4ファイルのサブミッションに分割するよう要求する
- C) 大きなコンテキストウィンドウを持つより高いティアのモデルに切り替えて、1回のパスで14ファイル全体を見る
- D) 3回の独立したフルPRレビューパスを実行し、少なくとも2回発見された問題のみを報告する

**Aの理由：** 集中したパスは根本原因—一度に多くのファイルを処理する際の注意力の希薄化—に直接対処します。ファイルごとの分析は一貫した深さを確保し、別の統合パスはクロスファイルの問題を検出します。Bは開発者に負担を移しシステムを改善しません。Cは誤解です：大きなコンテキストは注意の質を修正しません。Dは一貫性のない検出を通じた合意を要求することで実際のバグを抑制します。

---

# 練習テスト

> 4つのシナリオにわたる60問。形式と難易度は実際の試験と一致しています。

> または、これらの問題をインタラクティブなHTML形式で練習できます: 練習テスト (JA)

# シナリオ：マルチエージェント調査システム

---

## 問題1（シナリオ：マルチエージェント調査システム）

**状況：** ドキュメント分析エージェントが、重要なメトリクスについて2つの信頼できるソースが直接矛盾する統計を含んでいることを発見しました：政府レポートは40%の成長を述べ、業界分析は12%を述べています。両方のソースは信頼できるように見え、この不一致は調査の結論に大きな影響を与える可能性があります。ドキュメント分析エージェントはこの状況をどのように処理すべきでしょうか？

**最も効果的なアプローチは何ですか？**

- A) 信頼性のヒューリスティックを適用して最も正しいと思われる数値を選び、その値で分析を完了し、不一致についての脚注を追加する。
- B) 両方の数値を矛盾としてマークせずに分析出力に含め、統合エージェントがより広いコンテキストに基づいて使用するものを決定できるようにする。
- C) 分析を停止してコーディネーターに即座にエスカレーションし、継続する前にどのソースがより権威があるかを決定するよう求める。
- D) 両方の数値でソースの帰属を含む明示的な矛盾の注釈をつけて分析を完了し、合成に渡す前にコーディネーターにデータの調整を決定させる。**【正解】**

**Dの理由：** このアプローチは責任の分離を維持します：分析エージェントはブロックせずにコアワークを完了し、明確な帰属とともに両方の矛盾する値を保持し、より広いコンテキストを持つコーディネーターに調整を正しく渡します。

---

## 問題2（シナリオ：マルチエージェント調査システム）

**状況：** Web検索とドキュメント分析エージェントがタスクを完了し、結果をコーディネーターに返しました。統合された調査レポートを作成するための次のステップは何ですか？

**最も適切な次のステップは何ですか？**

- A) 各エージェントがコーディネーターをバイパスしてレポート作成エージェントに直接結果を送る。
- B) ドキュメント分析エージェントがWeb検索結果を要求し、内部で統合する。
- C) コーディネーターが両方の結果セットを統合エージェントに渡して一元的な統合を行う。**【正解】**
- D) コーディネーターが両方のエージェントからの生の出力を連結して最終結果として返す。

**Cの理由：** コーディネーター-サブエージェントアーキテクチャでは、コーディネーターが両方の結果セットを統合エージェントに転送して集中型の統合を行い、制御を維持し高品質なマージを確保します。

---

## 問題3（シナリオ：マルチエージェント調査システム）

**状況：** ドキュメント分析サブエージェントはPDFファイルの処理時に頻繁に失敗します：一部は破損したセクションがあり、他はパスワード保護されており、大きなファイルでは解析ライブラリがハングする場合があります。現在、例外が発生するとサブエージェントが即座に終了し、コーディネーターがエラーを返します。これにより定常的なエラー処理へのコーディネーターの過剰な関与が生じます。最も効果的なアーキテクチャの改善は何ですか？

**最も効果的な改善は何ですか？**

- A) 共有キューを介してすべての失敗を監視し、リカバリアクションを決定し、サブエージェントに直接再起動コマンドを送信する専用のエラーハンドリングエージェントを作成する。
- B) サブエージェントが常に成功ステータスで部分的な結果を返し、メタデータにエラーの詳細を埋め込むように設定する。コーディネーターはすべてのレスポンスを成功として扱う。
- C) コーディネーターがサブエージェントに送信する前にすべてのドキュメントを検証し、失敗を引き起こす可能性のあるドキュメントを拒否する。
- D) 一時的な失敗のサブエージェント内でのローカルリカバリを実装し、解決できないエラーのみを試みたステップと部分的な結果とともにコーディネーターにエスカレーションする。**【正解】**

**Dの理由：** エラーを解決できる最も低いレベルで処理します。ローカルリカバリはコーディネーターのワークロードを削減しながら、完全なコンテキストと部分的な進捗を持つ真に回復不可能な問題をエスカレーションします。

---

## 問題4（シナリオ：マルチエージェント調査システム）

**状況：** 「クリエイティブ産業へのAIの影響」でシステムを実行した後、すべてのサブエージェントが正常に完了することが観察されます：Web検索エージェントは関連する記事を見つけ、ドキュメント分析エージェントは正しく要約し、統合エージェントは一貫したテキストを生成します。しかし、最終レポートは視覚芸術のみをカバーし、音楽、文学、映画を完全に見落としています。コーディネーターのログでは、トピックが3つのサブタスクに分解されたことが確認できます：「デジタルアートにおけるAI」「グラフィックデザインにおけるAI」「写真撮影におけるAI」。最も可能性の高い根本原因は何ですか？

**最も可能性の高い根本原因は何ですか？**

- A) 統合エージェントにカバレッジのギャップを検出する指示がない。
- B) ドキュメント分析エージェントが過度に厳格な関連性基準により非視覚的なソースをフィルタリングしている。
- C) コーディネーターのタスク分解が過度に狭く、関連するすべての領域をカバーしていないサブエージェントに作業を割り当てている。**【正解】**
- D) Web検索エージェントのクエリが不十分で、より多くのセクターをカバーするために広げる必要がある。

**Cの理由：** コーディネーターが広いトピックを視覚芸術のサブタスクのみに分解し、音楽、文学、映画を完全に見落としました。サブエージェントは割り当てを正しく実行したため、狭い分解が明らかな根本原因です。

---

## 問題5（シナリオ：マルチエージェント調査システム）

**状況：** Web検索サブエージェントは5つの要求されたソースカテゴリのうち3つのみの結果を返します（競合サイトと業界レポートは成功しましたが、ニュースアーカイブとソーシャルフィードはタイムアウトしました）。ドキュメント分析サブエージェントは提供されたすべてのドキュメントを正常に処理します。統合サブエージェントは混合品質の上流入力からサマリーを生成する必要があります。最も効果的なエラー伝播戦略はどれですか？

**最も効果的なエラー伝播戦略は何ですか？**

- A) 成功したソースのみを使用して統合を継続し、どのデータが利用できなかったかを言及せずに出力を生成する。
- B) 統合サブエージェントはコーディネーターにエラーを返し、不完全なデータにより完全な再試行またはタスクの失敗をトリガーする。
- C) 統合サブエージェントはコーディネーターに統合を開始する前に、より長いタイムアウトでタイムアウトしたソースを再試行するよう依頼する。
- D) どの結論がよくサポートされており、利用できないソースによりどこにギャップがあるかを示すカバレッジ注釈で統合出力を構造化する。**【正解】**

**Dの理由：** カバレッジ注釈は透明性を持ったグレースフルデグラデーションを実装し、完了した作業の価値を保持しながら、信頼度について情報に基づく決定を可能にするために不確実性を伝播します。

---

## 問題6（シナリオ：マルチエージェント調査システム）

**状況：** ドキュメント分析サブエージェントが解析できない破損したPDFファイルに遭遇しました。システムのエラーハンドリングを設計する際、この失敗を処理する最も効果的な方法は何ですか？

**最も効果的なアプローチは何ですか？**

- A) コンテキストとともにエラーをコーディネーターエージェントに返し、続行方法を決定させる。**【正解】**
- B) ワークフローを中断しないよう破損したドキュメントをサイレントにスキップし、残りのファイルの処理を続ける。
- C) 失敗を報告する前に指数バックオフでドキュメントの解析を3回自動的に再試行する。
- D) 調査ワークフロー全体を終了する例外を投げる。

**Aの理由：** コンテキストとともにコーディネーターにエラーを返すことは最も効果的なアプローチです。コーディネーターがファイルのスキップ、代替解析方法の試み、またはユーザーへの通知など、情報に基づいた決定を行えるようにしながら、失敗の可視性を維持するからです。

---

## 問題7（シナリオ：マルチエージェント調査システム）

**状況：** 本番ログに一貫したパターンが示されています：「アップロードされた四半期レポートを分析して」などのリクエストが、ドキュメント分析エージェントの代わりに45%の割合でWeb検索エージェントにルーティングされています。ツール定義を確認すると、Web検索エージェントには「コンテンツを分析し主要情報を抽出する」と説明された`analyze_content`というツールがあり、ドキュメント分析エージェン

トには「ドキュメントを分析し主要情報を抽出する」と説明された`analyze_document`というツールがあります。誤ルーティングの問題をどのように修正すべきですか？

**誤ルーティングの問題をどのように修正すべきですか？**

- A) ユーザーがアップロードされたファイルを参照しているかWeb上のコンテンツを参照しているかを検出し、コーディネーターが委譲を決定する前に前処理分類器を追加する。
- B) Web検索ツールを`extract_web_results`に名前を変更し、その説明を「Web検索とURLから取得した情報を処理して返す」に更新する。**【正解】**
- C) コーディネーターのプロンプトに正しいルーティングを示すFew-shotサンプルを追加する：「ユーザーが四半期レポートをアップロードする → ドキュメント分析エージェント」と「ユーザーがWebページについて質問する → Web検索エージェント」。
- D) ドキュメント分析ツールの説明に「アップロードされたPDF、Wordドキュメント、スプレッドシートに使用する」などの使用例を追加し、Web検索ツールはそのままにする。

**Bの理由：** Web検索ツールを`extract_web_results`に名前変更し、説明をWeb検索とURLを明示的に参照するように更新することで、2つのツール名と説明の意味的な重複を排除し、根本原因に直接対処します。これにより各ツールの目的が明確になり、コーディネーターがドキュメント分析とWeb検索を確実に区別できるようになります。

---

## 問題8（シナリオ：マルチエージェント調査システム）

**状況：** 同僚がドキュメント分析エージェントはコーディネーターをバイパスして統合エージェントに直接結果を送るべきだと提案しています。サブエージェント間のすべての通信の中央ハブとしてコーディネーターを維持することの主な利点は何ですか？

**コーディネーターを中央ハブとして維持することの主な利点は何ですか？**

- A) コーディネーターはすべての相互作用を観察し、エラーを均一に処理し、各サブエージェントが受け取る情報を決定できる。**【正解】**
- B) コーディネーターはサブエージェントへの複数のリクエストをバッチ処理し、総API呼び出しと全体的なレイテンシを削減する。
- C) コーディネーターを経由したルーティングにより、直接のエージェント間呼び出しではサポートできない自動再試行ロジックが可能になる。
- D) サブエージェントは隔離されたメモリを使用し、直接通信には複雑なシリアライズが必要でコーディネーターだけが実行できる。

**Aの理由：** コーディネーターパターンはすべての相互作用への集中した可視性、システム全体での均一なエラーハンドリング、各サブエージェントが受け取る情報へのきめ細かい制御を提供します—これらはスター型の通信トポロジーの主な利点です。

---

## 問題9（シナリオ：マルチエージェント調査システム）

**状況：** Web検索サブエージェントが複雑なトピックを調査中にタイムアウトします。この失敗に関する情報をコーディネーターにどのように返すかを設計する必要があります。どのエラー伝播アプローチがインテリジェントなリカバリを最もよく可能にしますか？

**どのエラー伝播アプローチがインテリジェントなリカバリを最もよく可能にしますか？**

- A) 失敗タイプ、実行されたクエリ、部分的な結果、および潜在的な代替アプローチを含む構造化されたエラーコンテキストをコーディネーターに返す。**【正解】**
- B) サブエージェント内でタイムアウトをキャッチし、空の結果セットを成功としてマークして返す。
- C) サブエージェント内で指数バックオフの自動再試行を実装し、再試行を使い果たした後にのみ汎用の「検索利用不可」ステータスを返す。
- D) タイムアウト例外を直接トップレベルのハンドラーに伝播し、調査ワークフロー全体を終了する。

**Aの理由：** 失敗タイプ、実行されたクエリ、部分的な結果、代替アプローチを含む構造化されたエラーコンテキストを返すことで、コーディネーターはインテリジェントなリカバリの決定を行うために必要なすべてのもの（例：変更されたクエリで再試行するか、部分的な結果で続行するか）を得られます。インテリジェントなコーディネーションレベルの意思決定のために最大のコンテキストを保持します。

---

## 問題10（シナリオ：マルチエージェント調査システム）

**状況：** システム設計では、ドキュメント分析エージェントにURLでドキュメントをダウンロードできる汎用ツール`fetch_url`へのアクセスを与えました。本番ログはこのエージェントが頻繁に検索エンジンの結果ページをダウンロードしてアドホックなWeb検索を実行していることを示しており、これはWeb検索エージェントを通じてルーティングされるべき動作です。最も効果的な修正はどれですか？

**最も効果的な修正はどれですか？**

- A) `fetch_url`を、URLがドキュメント形式を指しているかどうかを検証する`load_document`ツールに置き換える。**【正解】**
- B) ドキュメント分析エージェントから`fetch_url`を削除し、すべてのURLフェッチをコーディネーターを通じてWeb検索エージェントにルーティングする。
- C) `fetch_url`呼び出しを既知の検索エンジンドメインへのものをブロックしながら他のURLを許可するフィルタリングを実装する。
- D) `fetch_url`はドキュメントURLのダウンロードにのみ使用し、検索には使用しないようにドキュメント分析エージェントのプロンプトに指示を追加する。

**Aの理由：** 汎用ツールをURLをドキュメント形式に対して検証するドキュメント固有のツールに置き換えることで、インターフェースレベルで機能を制限することによって根本原因を修正します。これは最小権限の原則に従い、望ましくない検索動作を単に思いとどまらせるのではなく不可能にします。

---

## 問題11（シナリオ：マルチエージェント調査システム）

**状況：** 広いトピックを調査する際、Web検索エージェントとドキュメント分析エージェントが同じサブトピックを調査し、出力に大幅な重複が生じます。調査の幅や深さの比例した増加なしにトークン使用量がほぼ倍増します。これに対処する最も効果的な方法は何ですか？

**最も効果的な方法は何ですか？**

- A) 両方のエージェントが並列で終了することを許可し、その後コーディネーターが重複する結果を排除してから統合エージェントに渡す。

- B) コーディネーターが委譲前に調査空間を明示的に分割し、各エージェントに異なるサブトピックまたはソースタイプを割り当てる。**【正解】**
- C) エージェントが現在のフォーカス領域を記録する共有状態メカニズムを実装し、他のエージェントが実行中に動的に重複を避けられるようにする。
- D) ドキュメント分析がWeb検索完了後にのみ実行される順次実行に切り替え、Web検索結果をコンテキストとして重複を避ける。

**Bの理由：** コーディネーターが委譲前に調査空間を明示的に分割することが最も効果的です。なぜなら、作業が始まる前に根本原因—不明確なタスク境界—に対処するからです。重複した作業や無駄なトークンを防ぎながら並列処理を保持します。

---

## 問題12（シナリオ：マルチエージェント調査システム）

**状況：** 調査中、Web検索サブエージェントは異なる結果で3つのソースカテゴリに対してクエリを実行します：学術データベースは15の関連論文を返し、業界レポートは「0件の結果」を返し、特許データベースは「接続タイムアウト」を返します。コーディネーターへのエラー伝播を設計する際、どのアプローチが最良のリカバリ決定を可能にしますか？

**どのアプローチが最良のリカバリ決定を可能にしますか？**

- A) 結果を単一の成功パーセンテージメトリクス（例：「67%のソースカバレッジ」）に集約し、詳細なログをオンデマンドで利用できるようにする。
- B) 「タイムアウト」と「0件の結果」の両方をコーディネーターの介入が必要な失敗として報告する。
- C) 内部で一時的な失敗を再試行し、永続的なエラーのみを報告する。
- D) リトライ決定が必要なアクセス失敗（タイムアウト）と成功したクエリを表す有効な空の結果（「0件の結果」）を区別する。**【正解】**

**Dの理由：** タイムアウト（アクセス失敗）と「0件の結果」（有効な空の結果）は異なる対応が必要な意味的に異なる結果です。これらを区別することで、コーディネーターは特許データベースを再試行しながら、業界レポートの「0件の結果」を有効な、情報価値のある発見として受け入れることができます。

---

## 問題13（シナリオ：マルチエージェント調査システム）

**状況：** 本番監視では統合品質が一貫していないことが示されています。集約された結果が約75Kトークンの場合、統合エージェントは最初の15Kトークン（Web検索の見出し/スニペット）と最後の10Kトークン（ドキュメント分析の結論）からの情報を確実に引用しますが、中間の50Kトークンの重要な発見を見落とすことが多く、それが調査の質問に直接答えるものであっても同様です。集約された入力をどのように再構築すべきですか？

**集約された入力をどのように再構築すべきですか？**

- A) 集約する前にすべてのサブエージェント出力を20K未満のトークンに要約し、コンテンツをモデルの信頼できる処理範囲内に保つ。
- B) サブエージェントの結果を統合エージェントに増分的にストリームし、Web検索結果を最初に完全に処理してからドキュメント分析結果を追加する。

- C) 集約された入力の先頭に重要な発見のサマリーを配置し、より簡単なナビゲーションのために明示的なセクション見出しで詳細な結果を整理する。**【正解】**
- D) 時間とともに両方のソースが等しくトップポジションを得るよう、どのサブエージェントの結果が最初に表示されるかをローテーションする。

**Cの理由：** 先頭に重要な発見サマリーを置くことで、重要な情報が最も確実に処理される位置に置かれる初頭効果を活用します。全体に明示的なセクション見出しを追加することで、モデルが中間のコンテンツをナビゲートして注意を向けるのを助け、「中間消失」現象に直接対処します。

---

## 問題14（シナリオ：マルチエージェント調査システム）

**状況：** テストでは、Web検索エージェント（ページコンテンツを含む85Kトークン）とドキュメント分析エージェント（思考の連鎖を含む70Kトークン）の組み合わせ出力が155Kトークンになりますが、統合エージェントは50K未満の入力で最もよくパフォーマンスを発揮します。最も効果的なソリューションはどれですか？

**最も効果的なソリューションはどれですか？**

- A) 上流エージェントを修正して、詳細なコンテンツや推論ではなく構造化データ（重要な事実、引用、関連性スコア）を返すようにする。**【正解】**
- B) 統合に渡す前に発見を凝縮する中間要約エージェントを追加する。
- C) 統合エージェントに発見を順次バッチで処理させ、呼び出し間で状態を維持する。
- D) 発見をベクターデータベースに保存し、統合エージェントが作業中にクエリを実行できる検索ツールを与える。

**Aの理由：** 上流エージェントを構造化データを返すように修正することで、重要な情報を保持しながらソースでのトークン量を削減して根本原因を修正します。統合ステップを改善せずにトークンを膨らませる大量のページコンテンツや推論トレースの受け渡しを避けます。

---

## 問題15（シナリオ：マルチエージェント調査システム）

**状況：** テストでは、統合エージェントが結果をマージしながら特定の主張を確認する必要があることが多く観察されます。現在、確認が必要な場合、統合エージェントはコーディネーターに制御を返し、コーディネーターはWeb検索エージェントを呼び出し、その後結果とともに統合を再起動します。これによりタスクごとに2〜3回の追加ループが追加され、レイテンシが40%増加します。評価では、これらの確認の85%は単純なファクトチェック（日付、名前、統計）で、15%はより深い調査が必要です。オーバーヘッドを最も効果的に削減しながらシステムの信頼性を維持するアプローチはどれですか？

**最も効果的なアプローチはどれですか？**

- A) 統合エージェントにすべてのWeb検索ツールへのアクセスを与え、コーディネーターのループなしにあらゆる確認ニーズを直接処理できるようにする。
- B) 統合エージェントがすべての確認ニーズを蓄積し、最後にバッチとしてコーディネーターに返し、コーディネーターがすべてをWeb検索エージェントに一度に送る。
- C) Web検索エージェントに初期調査中に各ソースの周りに追加のコンテキストをプロアクティブにキャッシュさせ、統合が確認を必要とすることを予測する。

- D) 単純なチェック向けに限定スコープの`verify_fact`ツールを統合エージェントに与え、複雑な確認はコーディネーター経由でWeb検索エージェントにルーティングする。**【正解】**

**Dの理由：** 限定スコープのファクト確認ツールにより、統合エージェントは85%の単純なチェックを直接処理し、ほとんどのループを排除しながら、15%の複雑な確認のコーディネーター委譲パスを維持します。これは最小権限を適用しながらレイテンシを大幅に削減します。

---

# シナリオ：Claude Code for Continuous Integration

---

# 問題16（シナリオ：Claude Code for Continuous Integration）

**状況：** CIパイプラインはCLAUDE.mdを使ってコードレビューのプロジェクトコンテキストを提供し、`--print`モードでClaude Code CLIを実行します。開発者は一般的にレビューが充実していると感じています。しかし、発見をワークフローに統合することが難しいと報告しています—ClaudeはPRコメントに手動でコピーする必要がある散文段落を出力します。チームはファイルパス、行番号、深刻度レベル、提案修正を含む構造化データを必要とする各発見をPRのコード内の関連箇所に別々のインラインPRコメントとして自動投稿したいです。最も効果的なアプローチはどれですか？

**最も効果的なアプローチはどれですか？**

- A) Claudeが期待される形式をプロジェクトコンテキストから学習するよう、構造化された発見のサンプルを含む「レビューの出力形式」セクションをCLAUDE.mdに追加する。
- B) `--output-format json`と`--json-schema` CLIフラグを使って構造化された発見を強制し、出力を解析してGitHub APIを通じてインラインコメントを投稿する。**【正解】**
- C) 各発見が`[FILE:path] [LINE:n] [SEVERITY:level] ...`のような解析可能なテンプレートに従うよう要求するレビュープロンプトに明示的な形式指示を含める。
- D) 散文のレビュー形式を維持し、Claudeを使って発見の構造化されたJSONサマリーを生成する要約ステップを追加する。

**Bの理由：** `--output-format json`と`--json-schema`を使用することで、CLIレベルでの構造化出力が強制され、GitHub APIを通じてインラインPRコメントとして確実に解析・投稿できる必要なフィールド（ファイルパス、行番号、深刻度、修正提案）を持つ適切な形式のJSONが保証されます。構造化出力のために特別に設計された組み込みのCLI機能を活用しています。

---

# 問題17（シナリオ：Claude Code for Continuous Integration）

**状況：** チームはClaude Codeを使ってコード提案を生成していますが、パターンに気づきます：エッジケースを壊すパフォーマンス最適化、予期せず動作を変更するクリーンアップなど、非明白な問題は別のチームメンバーがPRをレビューするときにのみ発見されます。生成中のClaudeの推論はこれらのケースを考慮したが、そのアプローチが正しいと結論付けたことを示しています。この自己チェックの限界の根本原因に直接対処するアプローチはどれですか？

**根本原因に直接対処するアプローチはどれですか？**

- A) ジェネレーターの推論にアクセスせずに変更をレビューするために2番目の独立したClaude Codeインスタンスを実行する。**【正解】**
- B) 生成段階に拡張思考モードを有効にして、提案を生成する前により徹底的な検討を可能にする。
- C) 最終出力を確定する前に自分の提案を批評するよう求める自己レビュー指示を生成プロンプトに追加する。
- D) 生成中にClaudeが期待される動作をよりよく理解できるよう、コンテキストにテストファイルと完全なドキュメントを含める。

**Aの理由：** ジェネレーターの推論にアクセスしない2番目の独立したClaude Codeインスタンスは、確証バイアスを避けることによって根本原因に直接対処します。この「新鮮な目」の視点は、著者が合理化した問題を別のレビュアーが見つける人間のピアレビューを反映しています。

---

## 問題18（シナリオ：Claude Code for Continuous Integration）

**状況：** コードレビューコンポーネントは反復的です：Claudeは変更されたファイルを分析し、最終フィードバックを提供する前にコンテキストを理解するために関連ファイル（インポート、基底クラス、テスト）をツール呼び出しで要求する場合があります。アプリケーションはClaudeがファイルの内容を要求できるツールを定義し、ClaudeはツールXを呼び出し、結果を取得し、分析を続けます。APIコストを削減するためにバッチ処理を評価しています。このワークフローでバッチ処理を検討する際の主要な技術的制限は何ですか？

**主要な技術的制限は何ですか？**

- A) バッチ処理には出力を入力リクエストにマッピングするための相関IDが含まれていない。
- B) 非同期モデルはリクエスト中にツールを実行し、Claudeが分析を継続するために結果を返すことができない。**【正解】**
- C) Batch APIはリクエストパラメータのツール定義をサポートしていない。
- D) バッチ処理のレイテンシは最大24時間かかりすぎてプルリクエストフィードバックには使えないが、ワークフロー自体はそれ以外では機能する。

**Bの理由：** 「ファイアアンドフォーゲット」型の非同期Batch APIモデルには、リクエスト中にツール呼び出しを傍受し、ツールを実行し、Claudeが分析を継続するために結果を返すメカニズムがありません。これは、1つの論理的なインタラクション内で複数のツールリクエスト/レスポンスラウンドを必要とする反復的なツール呼び出しワークフローと根本的に互換性がありません。

---

## 問題19（シナリオ：Claude Code for Continuous Integration）

**状況：** CI/CDシステムは3つのClaudeベースの分析を実行します：(1) マージが完了するまでブロックするすべてのPRに対する高速スタイルチェック、(2) コードベース全体の包括的な週次セキュリティ監査、(3) 最近変更されたモジュールの夜間テストケース生成。Message Batches APIは50%の節約を提供しますが、処理には最大24時間かかる場合があります。APIコストを最適化しながら許容可能な開発者エクスペリエンスを維持したいです。各タスクを正しくAPIアプローチに一致させる組み合わせはどれですか？

**どの組み合わせが正しいですか？**

- A) 50%節約を最大化するために3つのタスクすべてにMessage Batches APIを使用し、バッチ完了をポーリングするようにパイプラインを設定する。
- B) PRスタイルチェックには同期呼び出しを使用し、週次セキュリティ監査と夜間テスト生成にはMessage Batches APIを使用する。**【正解】**
- C) 一貫したレスポンス時間のために3つのタスクすべてに同期呼び出しを使用し、ワークロード全体でコストを削減するためにプロンプトキャッシングを利用する。
- D) PRスタイルチェックと夜間テスト生成には同期呼び出しを使用し、週次セキュリティ監査にのみMessage Batches APIを使用する。

**Bの理由：** PRスタイルチェックは開発者をブロックし、同期呼び出しによる即時レスポンスを必要としますが、週次セキュリティ監査と夜間テスト生成は最大24時間のバッチウィンドウを許容できる柔軟な期限を持つ予定されたタスクです—両方で50%の節約を実現できます。

---

## 問題20（シナリオ：Claude Code for Continuous Integration）

**状況：** 自動レビューは実際の問題を見つけますが、開発者はフィードバックが実用的でないと報告しています。発見には何を正確に変更するかを指定しない「複雑なチケットルーティングロジック」や「潜在的なnullポインター」などのフレーズが含まれています。「常に具体的な修正提案を含める」などの詳細な指示を追加しても、モデルは一貫性のない出力—時に詳細、時に曖昧—を生成します。一貫して実用的なフィードバックを最も確実に生成するプロンプティングテクニックはどれですか？

**最も確実なプロンプティングテクニックはどれですか？**

- A) フィードバック形式の各部分（場所、問題、深刻度、提案修正）に対してより明示的な要件を持つ指示をさらに洗練する。
- B) モデルが具体的な修正を提案するのに十分な情報を持つよう、より多くの周辺コードベースを含めるためにコンテキストウィンドウを拡大する。
- C) 1つのプロンプトが問題を特定し、2番目のプロンプトが修正を生成する2段階のアプローチを実装し、専門化を可能にする。
- D) 特定された問題、コード内の場所、具体的な修正提案などの必要な形式を示す3〜4のFew-shotサンプルを追加する。**【正解】**

**Dの理由：** Few-shotサンプルは指示だけでは変動する出力が生成される場合に一貫した出力形式を達成するための最も効果的な技術です。正確な望ましい構造（問題、場所、具体的な修正）を示す3〜4のサンプルを提供することで、抽象的な指示よりも信頼性の高いパターンをモデルに与えます。

---

## 問題21（シナリオ：Claude Code for Continuous Integration）

**状況：** CIパイプラインには2つのClaudeベースのコードレビューモードがあります：マージを完了するまでブロックするPRプリマージコミットフック、および夜間に実行され、バッチ完了をポーリングし、詳細な提案をPRに投稿する「深い分析」。50%の節約を提供するが最大24時間かかる可能性があるMessage Batches APIを使ってAPIコストを削減したいです。どのモードがバッチ処理を使うべきですか？

**どのモードがバッチ処理を使うべきですか？**

- A) PRプリマージコミットフックのみ。

- B) 深い分析のみ。**【正解】**
- C) 両方のモード。
- D) どちらのモードも使わない。

**Bの理由：** 深い分析は夜間に実行され、遅延を許容し、結果を公開する前のポーリングモデルを使用しているため、バッチ処理の理想的な候補です—Message Batches APIの非同期ポーリングベースのアーキテクチャに合致しながら50%の節約を実現します。

---

## 問題22（シナリオ：Claude Code for Continuous Integration）

**状況：** 自動レビューはコメントとdocstringを分析します。現在のプロンプトはClaudeに「コメントが正確で最新であることを確認する」よう指示します。発見は許容可能なパターン（TODOマーカー、単純な説明）にフラグを立てる一方、コードが実装しなくなった動作を説明するコメントを見落とすことが多いです。この不一致な分析の根本原因に対処する変更はどれですか？

**根本原因に対処する変更はどれですか？**

- A) Claudeが最近のコード変更より前のコメントを特定できるように`git blame`データを含める。
- B) モデルがコードベース内の同様のパターンを認識するのを助けるために、誤解を招くコメントのFew-shotサンプルを追加する。
- C) ノイズを減らすために分析前にTODO、FIXME、説明的なコメントパターンをフィルタリングする。
- D) 明示的な基準を指定する：コメントが主張する動作が実際のコードの動作と矛盾する場合にのみコメントにフラグを立てる。**【正解】**

**Dの理由：** 明示的な基準—主張された動作が実際のコード動作と矛盾する場合にのみコメントにフラグを立てる—は、曖昧な指示を問題の正確な定義に置き換えることで根本原因に直接対処します。これにより許容可能なパターンでの偽陽性が減り、本当に誤解を招くコメントの見落としが減ります。

---

## 問題23（シナリオ：Claude Code for Continuous Integration）

**状況：** 自動コードレビューシステムは一貫性のない深刻度評価を示しています—nullポインターリスクのような類似した問題が、あるPRでは「critical」と評価され、別のPRでは「medium」としか評価されません。開発者調査では不信感が高まっています—多くが「半分は間違っている」ために読まずに発見を無視し始めています。高い偽陽性カテゴリは正確なカテゴリへの信頼を損なっています。開発者の信頼を回復しながらシステムを改善する最善のアプローチはどれですか？

**開発者の信頼を最も回復するアプローチはどれですか？**

- A) 高偽陽性カテゴリ（スタイル、ネーミング、ドキュメント）を一時的に無効化し、プロンプトを改善しながら高精度カテゴリのみを維持する。**【正解】**
- B) すべてのカテゴリを有効にしたままにして、開発者が調査するものを決定できるよう各発見に信頼度スコアを表示する。
- C) すべてのカテゴリを有効にしたままにして、次の数週間で各カテゴリの精度を向上させるためにFew-shotサンプルを追加する。

- D) すべてのカテゴリに均一な厳格さの削減を適用して全体的な偽陽性率を下げる。

**Aの理由：** 高偽陽性カテゴリを一時的に無効化することで、開発者がすべてを無視する原因となるノイズの多い発見を除去し、信頼の侵食を即座に停止しながら、セキュリティや正確性などの高精度カテゴリの価値を保持します。また、再有効化する前に問題のあるカテゴリのプロンプトを改善するための余地を作ります。

---

## 問題24（シナリオ：Claude Code for Continuous Integration）

**状況：** 自動レビューは各PRにテストケースの提案を生成します。コース完了トラッキングを追加するPRをレビューする際、Claudeは10のテストケースを提案しますが、開発者のフィードバックでは6つが既存のテストスイートでカバーされているシナリオと重複しています。重複した提案を最も効果的に削減する変更はどれですか？

**最も効果的な変更はどれですか？**

- A) Claudeが既にカバーされているシナリオを判断できるように、コンテキストに既存のテストファイルを含める。**【正解】**
- B) 要求する提案の数を10から5に減らし、Claudeが最も価値の高いケースを最初に優先すると仮定する。
- C) Claudeが成功パスではなくエッジケースとエラー状態のみに集中するよう指示を追加する。
- D) キーワードの重複によって説明が既存のテスト名と一致する提案をフィルタリングする後処理を実装する。

**Aの理由：** 既存のテストファイルを含めることで、重複の根本原因を修正します：Claudeは既にどのテストが存在するかを知っている場合にのみ、既にカバーされているシナリオの提案を避けることができます。これにより、Claudeが真に新しく価値のあるテストを提案するために必要な情報が提供されます。

---

## 問題25（シナリオ：Claude Code for Continuous Integration）

**状況：** 最初の自動レビューで12の発見が特定された後、開発者は問題に対処するために新しいコミットをプッシュします。レビューを再実行すると8の発見が生成されますが、開発者は新しいコミットで既に修正されたコードに対する5つの以前のコメントの重複を報告します。徹底性を維持しながらこの冗長なフィードバックを排除する最も効果的な方法はどれですか？

**最も効果的な方法はどれですか？**

- A) PRが作成されたときと最終的なプリマージ状態にのみレビューを実行し、中間のコミットをスキップする。
- B) ファイルパスと問題の説明によって以前のものと一致する発見を投稿前に除去する後処理フィルターを追加する。
- C) レビュースコープを最近のプッシュで変更されたファイルに制限し、以前のコミットからのファイルを除外する。
- D) 以前のレビュー発見をコンテキストに含め、新規または未解決の問題のみを報告するようClaudeに指示する。**【正解】**

**Dの理由：** コンテキストに以前のレビュー発見を含めることで、Claudeが最近のコミットで既に対処された問題と新しい問題を区別できるようにします。これにより修正されたコードへの冗長なフィードバックを避けながら、Claudeの推論を使ってレビューの徹底性を維持します。

---

## 問題26（シナリオ：Claude Code for Continuous Integration）

**状況：** パイプラインスクリプトが`claude "このプルリクエストのセキュリティ問題を分析してください"`を実行しますが、ジョブが無期限にハングします。ログはClaude Codeがインタラクティブな入力を待っていることを示しています。自動化されたパイプラインでClaude Codeを実行する正しいアプローチは何ですか？

**正しいアプローチは何ですか？**

- A) `--batch`フラグを追加する：`claude --batch "このプルリクエストのセキュリティ問題を分析してください"`。
- B) `-p`フラグを追加する：`claude -p "このプルリクエストのセキュリティ問題を分析してください"`。**【正解】**
- C) `/dev/null`からstdinをリダイレクトする：`claude "このプルリクエストのセキュリティ問題を分析してください" < /dev/null`。
- D) コマンドを実行する前に環境変数`CLAUDE_HEADLESS=true`を設定する。

**Bの理由：** `-p`（または`--print`）フラグはClaude Codeを非インタラクティブで実行するための文書化された方法です。プロンプトを処理し、結果をstdoutに出力し、ユーザー入力を待たずに終了します—CI/CDパイプラインに最適です。

---

## 問題27（シナリオ：Claude Code for Continuous Integration）

**状況：** プルリクエストが在庫追跡モジュールの14ファイルを変更しています。すべてのファイルを一緒に分析する単一パスのレビューは一貫性のない結果を生成します：一部のファイルには詳細なフィードバック、他のファイルには浅いコメント、明らかなバグの見落とし、矛盾したフィードバック（同じPR内の別のファイルでは同一コードが承認されているのに、1つのファイルでパターンにフラグが立てられる）。レビューをどのように再構築すべきですか？

**どのように再構築すべきですか？**

- A) 3つの独立したフルPRレビューパスを実行し、少なくとも2つの実行で現れる問題のみにフラグを立てる。
- B) ローカルの問題について各ファイルを個別にレビューする集中したパスに分割し、クロスファイルのデータフローを調査する別の統合指向のパスを実行する。**【正解】**
- C) 開発者に自動レビューを実行する前に大きなPRを3〜4ファイルの小さなサブミッションに分割するよう要求する。
- D) 1回のパスで14ファイルすべてに十分な注意を払えるよう、より大きなコンテキストウィンドウを持つ大きなモデルに切り替える。

**Bの理由：** 集中したファイルごとのパスは根本原因—注意力の希薄化—に対処し、一貫した深さと信頼できるローカルの問題検出を確保します。別の統合指向のパスは、依存関係やデータフローの相互作用などのクロスファイルの懸念事項をカバーします。

---

## 問題28（シナリオ：Claude Code for Continuous Integration）

**状況：** 自動コードレビューはPRごとに平均15の発見を示し、開発者は40%の偽陽性率を報告しています。ボトルネックは調査時間です：開発者は修正するか無視するかを決める前に各発見をクリックしてClaudeの根拠を読む必要があります。CLAUDE.mdには許容可能なパターンに対する包括的なルールがすでに含まれており、利害関係者は開発者が見る前に発見をフィルタリングするアプローチを拒否しました。調査時間に最も対処する変更はどれですか？

**調査時間に最も対処する変更はどれですか？**

- A) 各発見に根拠と信頼度の見積もりを直接含めるようClaudeに要求する。**【正解】**
- B) 発見パターンを分析し、歴史的な偽陽性のシグネチャに一致するものを自動的に抑制する後処理を追加する。
- C) 発見を「ブロッキング問題」と「提案」に分類し、レベルごとに異なるレビュー要件を持つ。
- D) 全体的な偽陽性率を下げるために確実でないフラグを開発者が見る前にフィルタリングして、高信頼度の発見のみを表示するようClaudeを設定する。

**Aの理由：** 各発見に根拠と信頼度を直接含めることで、開発者が各発見を開くことなく迅速にトリアージできるようにし、調査時間を削減します。すべての発見が表示されたままであるため「フィルタリングなし」の制約を満たしながら、開発者の意思決定を加速します。

---

## 問題29（シナリオ：Claude Code for Continuous Integration）

**状況：** 自動コードレビューの分析では、発見カテゴリ別に大きな偽陽性率の違いが示されています：セキュリティ/正確性の発見は8%の偽陽性、パフォーマンスは18%、スタイル/ネーミングは52%、ドキュメントは48%。開発者調査では不信感が高まっています—多くが「半分は間違っている」ために読まずに発見を無視し始めています。高偽陽性カテゴリは正確なカテゴリへの信頼を損なっています。開発者の信頼を最も回復しながらシステムを改善するアプローチはどれですか？

**開発者の信頼を最も回復するアプローチはどれですか？**

- A) 高偽陽性カテゴリ（スタイル、ネーミング、ドキュメント）を一時的に無効化し、プロンプトを改善しながら高精度カテゴリのみを維持する。**【正解】**
- B) すべてのカテゴリを有効にしたままにして、開発者が調査するものを決定できるよう各発見に信頼度スコアを表示する。
- C) すべてのカテゴリを有効にしたままにして、次の数週間で各カテゴリの精度を向上させるためにFew-shotサンプルを追加する。
- D) すべてのカテゴリに均一な厳格さの削減を適用して全体的な偽陽性率を下げる。

**Aの理由：** 高偽陽性カテゴリを一時的に無効化することで、開発者がすべてを無視する原因となるノイズの多い発見を除去し、信頼の侵食を即座に停止しながら、セキュリティや正確性などの高精度カテゴリの

価値を保持します。また、再有効化する前に問題のあるカテゴリのプロンプトを改善するための余地を作ります。

---

## 問題30（シナリオ：Claude Code for Continuous Integration）

**状況：** チームはAPIコストを削減したいです。現在、同期Claude呼び出しが2つのワークフローをサポートします：(1) 開発者がマージできるようになる前に完了する必要があるブロッキングプリマージチェック、(2) 翌朝のレビューのために夜間に生成される技術的負債レポート。マネージャーは50%節約のためにMessage Batches APIに両方を移行することを提案しています。この提案をどのように評価すべきですか？

**どのように評価すべきですか？**

- A) バッチに時間がかかりすぎる場合に同期呼び出しへのフォールバックで両方をバッチ処理に移行する。
- B) ステータスポーリングで完了を確認して両方のワークフローをバッチ処理に移行する。
- C) 技術的負債レポートにのみバッチ処理を使用し、プリマージチェックには同期呼び出しを維持する。**【正解】**
- D) バッチ結果の順序付けの問題を避けるために両方のワークフローに同期呼び出しを維持する。

**Cの理由：** Message Batches APIの処理は最大24時間かかる可能性があり、レイテンシSLAの保証がありません。これは開発者が待つブロッキングプリマージチェックには許容できませんが、技術的負債レポートのような夜間バッチワークロードには理想的です。各ワークフローをレイテンシ要件に基づいて適切なAPIに一致させます。

---

## シナリオ：Claude Codeによるコード生成

---

## 問題31（シナリオ：Claude Codeによるコード生成）

**状況：** Claudeに、APIレスポンスを内部の正規化された形式に変換する関数を実装するよう依頼しました。2回の反復後も、出力構造が期待と一致しません—一部のフィールドは異なるネスト方法で、タイムスタンプの形式が間違っています。要件を散文で説明しましたが、Claudeは毎回異なる解釈をします。

**次の反復で最も効果的なアプローチはどれですか？**

- A) 予想される出力構造を記述するJSONスキーマを作成し、各反復後にClaudeの出力を検証する。
- B) 代表的なAPIレスポンスの予想される変換を示す2〜3の具体的な入力-出力サンプルを提供する。**【正解】**
- C) 正確なフィールドマッピング、ネストルール、タイムスタンプ形式文字列を指定してより技術的な精度で要件を書き直す。
- D) 解釈が分岐している箇所を特定するために要件の現在の理解をClaudeに説明させる。

**Bの理由：** 具体的な入力-出力サンプルは、フィールドのネストとタイムスタンプ形式のための明確なパターンを提供することで、散文の説明に固有の曖昧さを排除します。これは根本原因—テキスト要件の誤

解—を直接解決します。

---

## 問題32（シナリオ：Claude Codeによるコード生成）

**状況：** 新しいSlack通知チャネルを追加する必要があります。既存のコードベースにはメール、SMS、プッシュチャネルの明確なパターンがあります。しかし、SlackのAPIは根本的に異なる統合アプローチを提供しています—インカミングWebhook（シンプルな一方向）、ボットトークン（配信確認とプログラム制御をサポート）、またはSlack App（双方向イベント、ワークスペースの承認が必要）。タスクには統合方法や高度な機能（配信追跡など）の要件が指定されていません。

**このタスクにどうアプローチすべきですか？**

- A) 既存の一方向通知パターンに合わせてインカミングWebhookを使用して直接実行モードで開始する。
- B) 計画モードに切り替えて統合オプションとアーキテクチャ的な影響を探索し、実装前に推奨事項を提示する。**【正解】**
- C) 既存のパターンを使ってSlackチャネルクラスの足場を作ることで直接実行モードで開始し、統合方法の決定を後回しにする。
- D) ボットトークンアプローチを使って直接実行モードで開始し、配信確認が可能なことを確保する。

**Bの理由：** Slack統合には大きく異なるアーキテクチャ的影響を持つ複数の有効なアプローチがあり、要件が曖昧です。計画モードにより、Webhook、ボットトークン、Slack Appのトレードオフを評価し、実装前にアプローチに合意できます。

---

## 問題33（シナリオ：Claude Codeによるコード生成）

**状況：** CLAUDE.mdファイルは400行以上に成長し、コーディング標準、テスト規約、詳細なPRレビューチェックリスト、デプロイ手順、データベースマイグレーション手順が含まれています。コーディング標準とテスト規約は常に適用させたいですが、PRレビュー、デプロイ、マイグレーションのガイダンスはそれらのタスクを実行するときのみ適用させたいです。

**最も効果的な再構築アプローチはどれですか？**

- A) すべてのガイダンスをワークフロータイプ別に整理された別のスキルファイルに移動し、CLAUDE.mdには簡単なプロジェクト説明のみを残す。
- B) CLAUDE.mdにすべてを維持したまま、カテゴリ別に別々に管理されたファイルに整理するための `@import` 構文を使用する。
- C) CLAUDE.mdをパスバインドされたglobパターンを持つ `.claude/rules/` の下のファイルに分割し、ファイルパスに基づいて関連するときのみ各ルールをロードする。
- D) CLAUDE.mdにユニバーサル標準を維持し、トリガーキーワードを持つワークフロー固有のガイダンス（PRレビュー、デプロイ、マイグレーション）のためのスキルを作成する。**【正解】**

**Dの理由：** CLAUDE.mdのコンテンツはすべてのセッションでロードされ、コーディング標準とテスト規約が常に適用されることを確保する一方、スキルはClaudeがトリガーキーワードを検出したときにオンデマンドで呼び出されます—PRレビュー、デプロイ、マイグレーションのようなワークフロー固有のガイダンスに最適です。

## 問題34（シナリオ：Claude Codeによるコード生成）

**状況：** チームのモノリシックアプリケーションをマイクロサービスに再構築するタスクを担当しています。これは数十のファイルにわたる変更とサービス境界やモジュール依存関係についての決定を必要とします。

**どのアプローチを選択すべきですか？**

- A) 計画モードに切り替えてコードベースを探索し、依存関係を理解し、変更を加える前に実装アプローチを設計する。**【正解】**
- B) 直接実行モードで開始し、実装中に予期しない複雑さに遭遇した後にのみ計画に切り替える。
- C) 直接実行モードで開始し、漸進的な変更を加えながら実装が自然なサービス境界を明らかにするようにする。
- D) 各サービス構造を指定する詳細な事前指示で直接実行を使用する。

**Aの理由：** 計画モードはモノリスの分割のような複雑なアーキテクチャの再構築に適した戦略です：多くのファイルにまたがる高価な変更にコミットする前に、安全な探索と境界についての情報に基づいた決定を可能にします。

---

## 問題35（シナリオ：Claude Codeによるコード生成）

**状況：** チームは深いコード分析を実行する`/analyze-codebase`スキルを作成しました—依存関係スキャン、テストカバレッジカウント、コード品質メトリクス。コマンドを実行した後、チームメンバーはClaudeがセッション内でレスポンスが遅くなり、元のタスクのコンテキストを失うと報告します。

**全分析機能を維持しながら最も効果的にこれを修正する方法はどれですか？**

- A) 隔離されたサブエージェントコンテキストで分析を実行するために、スキルのフロントマターに`context: fork`を追加する。**【正解】**
- B) フロントマターに`model: haiku`を追加して分析に高速で安価なモデルを使用する。
- C) スキルをそれぞれより少ない出力を生成する3つの小さなスキルに分割する。
- D) 表示前にすべての結果を短い要約に圧縮するようスキルに指示を追加する。

**Aの理由：** `context: fork`は隔離されたサブエージェントコンテキストで分析を実行するため、大量の出力がメインセッションのコンテキストウィンドウを汚染せず、Claudeが元のタスクを見失わなくなります。完全な分析機能を維持しながらメインセッションのレスポンスを維持します。

---

## 問題36（シナリオ：Claude Codeによるコード生成）

**状況：** チームは`.claude/skills/commit/SKILL.md`に`/commit`スキルを使用しています。ある開発者がチームメンバーに影響を与えずに個人的なワークフロー（異なるコミットメッセージ形式、追加のチェック）のためにカスタマイズしたいと考えています。

**何を推奨しますか？**

- A) `~/.claude/skills/`の下に`/my-commit`などの別の名前で個人バージョンを作成する。**【正解】**
- B) プロジェクトスキルのフロントマターにユーザー名に基づいた条件ロジックを追加する。
- C) 同じ名前で`~/.claude/skills/commit/SKILL.md`に個人バージョンを作成する。
- D) 個人スキルのフロントマターに`override: true`を設定してプロジェクトバージョンよりも優先させる。

**Aの理由：** 個人スキルは同じ名前のプロジェクトスキルよりも優先されるため、`commit`という名前を再利用すると、その開発者だけに対してチームのスキルが静かに隠されてしまいます。チームが`/commit`を改善しても更新を受け取れなくなり、同じコマンド名で別のスキルを実行していることを本人が覚えておく必要があります。個人用のバリアントを`/my-commit`と名付ければ、この衝突を完全に回避できます。開発者はチームが管理する`/commit`を引き続き使用しつつ、個人のワークフロー用に明確に別名の付いたスキルを持つことができ、両者を混同したりチームの更新を見逃したりする心配がありません。

---

## 問題37（シナリオ：Claude Codeによるコード生成）

**状況：** チームは数ヶ月間Claude Codeを使用しています。最近、3人の開発者は「包括的なエラーハンドリングを含める」というガイダンスにClaudeが従うと報告していますが、先週参加した4番目の開発者はClaudeが従わないと言っています。4人全員が同じリポジトリで作業し、最新のコードを持っています。

**最も可能性の高い原因と修正は何ですか？**

- A) ガイダンスはプロジェクトの`.claude/CLAUDE.md`ではなく元の開発者のユーザーレベル`~/.claude/CLAUDE.md`ファイルにあります。すべてのチームメンバーが受け取れるようにプロジェクトレベルのファイルに指示を移動してください。**【正解】**
- B) 新しい開発者の`~/.claude/CLAUDE.md`にはプロジェクト設定を上書きする矛盾した指示が含まれています。競合するセクションを削除すべきです。
- C) Claude Codeはユーザーごとの好みを時間とともに学習します。新しい開発者はClaudeが「記憶」するまで要件を繰り返す必要があります。
- D) Claude CodeはCLAUDE.mdを最初の読み込みの後にキャッシュします。元の開発者はキャッシュされたバージョンを使用しています。全員がClaude Codeキャッシュをクリアすべきです。

**Aの理由：** ガイダンスが元の開発者のユーザーレベル設定にのみ追加され、プロジェクトレベルの`.claude/CLAUDE.md`には追加されなかった場合、新しいチームメンバーはそれを受け取りません。プロジェクトレベルの設定に移動することで、現在および将来のすべてのチームメンバーが自動的にガイダンスを受け取れるようになります。

---

## 問題38（シナリオ：Claude Codeによるコード生成）

**状況：** コンテキストとして2〜3の完全なエンドポイント実装サンプルを含めることで、新しいAPIエンドポイントを生成する際の一貫性が大幅に向上することがわかりました。しかし、このコンテキストは新しいエンドポイントを作成する場合にのみ有用で、デバッグ、コードレビュー、またはAPIディレクトリの他の作業には有用ではありません。

**最も効果的な設定アプローチはどれですか？**

- A) 常に利用できるようにプロジェクトのCLAUDE.mdにエンドポイントのサンプルとパターンドキュメントを追加する。
- B) プロンプトにコードをコピーすることで、すべての生成リクエストでエンドポイントのサンプルを手動で参照する。
- C) APIディレクトリで作業する際に有効化されるエンドポイントのサンプルとパターン遵守指示を含む`.claude/rules/api/`のパス固有のルールを設定する。
- D) エンドポイントのサンプルを参照し、パターン遵守指示を含むスキルを作成し、スラッシュコマンドでオンデマンドで呼び出す。**【正解】**

**Dの理由：** オンデマンドで呼び出されるスキルは、デバッグやレビューのような関係のないタスク中ではなく、新しいエンドポイントを生成する場合にのみサンプルコンテキストをロードします。これにより必要なときに高品質な生成を維持しながらメインのコンテキストをクリーンに保ちます。

---

## 問題39（シナリオ：Claude Codeによるコード生成）

**状況：** チームはデータベースのマイグレーションファイルを生成する`/migration`スキルを作成しました。`$ARGUMENTS`でマイグレーション名を受け取ります。本番環境では3つの問題が観察されます：(1) 開発者がしばしば引数なしでスキルを実行し、名前の付け方が悪いファイルが生成される、(2) スキルが時々無関係な以前の会話からのデータベーススキーマの詳細を使用する、(3) 開発者がスキルに広いツールアクセスがあったとき、誰かが偶然に破壊的なテストのクリーンアップを実行した。3つの問題すべてを修正する設定アプローチはどれですか？

**3つの問題すべてを修正する設定アプローチはどれですか？**

- A) 特定の入力を強制するために`$ARGUMENTS`の代わりに位置パラメータ`$1`と`$2`を使用し、コンテキスト制御のために`@`構文を通じて明示的なスキーマファイル参照を含め、破壊的な操作についての警告を含むフロントマターの説明を追加する。
- B) 必要なパラメータを要求するためにフロントマターに`argument-hint`を追加し、実行を隔離するために`context: fork`を使用し、ファイル書き込み操作に`allowed-tools`を制限する。**【正解】**
- C) `/migration-create`と`/migration-apply`スキルに分割し、マイグレーション名が欠けている場合に要求するバリデーション指示を追加し、それぞれに異なる`allowed-tools`スコープを使用する。
- D) `$ARGUMENTS`が有効な名前であることを確認するためのバリデーション指示をスキルのSKILL.mdに追加し、以前の会話コンテキストを無視するプロンプトを追加し、避けるべき禁止操作をリストする。

**Bの理由：** これは各問題に対処するために3つの別々の設定機能を使用します：`argument-hint`は引数入力を改善し引数の欠如を減らし、`context: fork`は以前の会話からのコンテキストの漏洩を防ぎ、`allowed-tools`はスキルを安全なファイル書き込み操作に制限し、破壊的なアクションを防ぎます。

---

## 問題40（シナリオ：Claude Codeによるコード生成）

**状況：** コードベースには異なるコーディング規約を持つ領域があります：Reactコンポーネントはhooksを使ったファンクショナルスタイルを使用し、APIハンドラーは特定のエラーハンドリングを含むasync/awaitを使用し、データベースモデルはリポジトリパターンに従います。テストファイルはコードの

隣にコードベース全体に分散しています（例：`Button.tsx`の隣に`Button.test.tsx`）。そして、場所に関係なくすべてのテストが同じ規約に従ってほしいです。

**Claudeがコードを生成する際に正しい規約を自動的に適用するための最もサポートされた方法は何ですか？**

- A) ルートCLAUDE.mdに各領域の見出しを付けてすべての規約を置き、どのセクションが適用されるかをClaudeが推測するよう任せる。
- B) 各コードタイプのスキルを`.claude/skills/`に作成し、各SKILL.mdに規約を埋め込む。
- C) それぞれの規約を含む別のCLAUDE.mdファイルを各サブディレクトリに置く。
- D) ファイルパスに基づいて規約を条件付きで適用するためのglobパターンを指定するYAMLフロントマターを持つルールファイルを`.claude/rules/`の下に作成する。**【正解】**

**Dの理由：** YAMLフロントマターとglobパターン（例：`**/*.test.tsx`、`src/api/**/*.ts`）を使った`.claude/rules/`ファイルは、ディレクトリ構造に関係なくパスベースの規約の確定的な適用を可能にします。これは分散したテストファイルのようなクロスカッティングパターンに最もサポートされたアプローチです。

---

# 問題41（シナリオ：Claude Codeによるコード生成）

**状況：** チームの標準コードレビューチェックリストを実行するカスタムスラッシュコマンド`/review`を作成したいです。リポジトリをクローンまたは更新したときにすべての開発者が利用できるようにしたいです。

**コマンドファイルはどこに作成すべきですか？**

- A) 各開発者のホームディレクトリの`~/.claude/commands/`に。
- B) プロジェクトリポジトリの`.claude/commands/`の下に。**【正解】**
- C) コマンドの配列として`.claude/config.json`に。
- D) ルートプロジェクトの`CLAUDE.md`に。

**Bの理由：** プロジェクトリポジトリの`.claude/commands/`の下にカスタムスラッシュコマンドを置くことで、バージョン管理され、リポジトリをクローンまたは更新したすべての開発者が自動的に利用できるようになります。これはClaude Codeのプロジェクトレベルのカスタムコマンドの意図された場所です。

---

# 問題42（シナリオ：Claude Codeによるコード生成）

**状況：** チームのCLAUDE.mdはTypeScript規約、テストガイダンス、APIパターン、デプロイ手順が混在して500行を超えています。開発者は適切なセクションを見つけて更新するのが難しいと感じています。

**Claude Codeがプロジェクトレベルの指示をトピック別のモジュールに整理するためにサポートするアプローチは何ですか？**

- A) CLAUDE.md内の特定のセクションにファイルパターンをマッピングする`.claude/config.yaml`を定義する。

- B) トピック別（例：`testing.md`、`api-conventions.md`）に1つのトピックをカバーする別のMarkdownファイルを`.claude/rules/`に作成する。**【正解】**
- C) Claudeが自動的に指示としてロードする関連サブディレクトリのREADME.mdファイルに指示を分割する。
- D) ディレクトリツリーの異なるレベルに複数のCLAUDE.mdファイルを作成し、それぞれが親の指示を上書きする。

**Bの理由：** Claude Codeは`.claude/rules/`ディレクトリをサポートし、そこにトピック別のガイダンス（例：`testing.md`、`api-conventions.md`）のための別のMarkdownファイルを作成でき、チームが大きな指示セットを集中管理されたモジュールに整理できます。

---

## 問題43（シナリオ：Claude Codeによるコード生成）

**状況：** チームが実装アプローチを選択する前にブレインストーミングと評価に使用するカスタムスキル`/explore-alternatives`を作成しました。スキルを実行した後、その後のClaudeのレスポンスが代替案の議論の影響を受けることがあります—拒否されたアプローチを参照したり、実際の実装を妨げる探索コンテキストを保持したりします。

**このスキルを最も効果的に設定するにはどうすればよいですか？**

- A) 探索ロジックをbashサブプロセスとして実行するために、スキルで`!`プレフィックスを使用する。
- B) スキルのフロントマターに`context: fork`を追加する。**【正解】**
- C) 探索コンテキストを廃棄すべき境界をマークするための2つのスキル—`/explore-start`と`/explore-end`—に分割する。
- D) `.claude/skills/`ではなく`~/.claude/skills/`にスキルを作成する。

**Bの理由：** `context: fork`は隔離されたサブエージェントコンテキストでスキルを実行するため、探索の議論がメインの会話履歴を汚染しません。これにより拒否されたアプローチやブレインストーミングのコンテキストが後続の実装作業に影響を与えることを防ぎます。

---

## 問題44（シナリオ：Claude Codeによるコード生成）

**状況：** チームはClaude Codeを通じてPRを検索しCIステータスを確認するためのGitHub MCPサーバーを追加したいです。6人の開発者それぞれが独自の個人GitHubアクセストークンを持っています。バージョン管理にクレデンシャルをコミットせずに、チーム全体で一貫したツールを提供したいです。

**最も効果的な設定アプローチはどれですか？**

- A) 各開発者が`claude mcp add --scope user`でユーザースコープでサーバーを追加する。
- B) `.env`ファイルからトークンを読み取りGitHub API呼び出しをプロキシするMCPサーバーラッパーを作成し、そのラッパーをプロジェクトの`.mcp.json`に追加する。
- C) 認証に環境変数の置換（`${GITHUB_TOKEN}`）を使ったプロジェクト`.mcp.json`にサーバーを追加し、プロジェクトのREADMEに必要な環境変数を文書化する。**【正解】**
- D) プレースホルダートークンでプロジェクトスコープでサーバーを設定し、ローカル設定で上書きするよう開発者に伝える。

**Cの理由：** 環境変数の置換を使ったプロジェクト`.mcp.json`は慣用的なアプローチです：シークレットをコミットせずに各開発者が環境変数でクレデンシャルを提供できるようにしながら、MCP設定のバージョン管理された単一のソースオブトゥルースを提供します。変数を文書化することでオンボーディングが簡単になります。

---

## 問題45（シナリオ：Claude Codeによるコード生成）

**状況：** 120ファイルのコードベース全体で外部API呼び出しにエラーハンドリングラッパーを追加しています。作業には3つのフェーズがあります：(1) すべての呼び出しサイトとパターンを発見する、(2) エラーハンドリングアプローチを共同設計する、(3) 一貫してラッパーを実装する。フェーズ1では、Claudeが数百の呼び出しサイトとコンテキストをリストする大きな出力を生成し、発見が完了する前にすぐにコンテキストウィンドウを満たしてしまいます。

**実装の一貫性を維持しながらタスクを完了する最も効果的なアプローチはどれですか？**

- A) フェーズ1で探索サブエージェントを使って詳細な発見出力を隔離してサマリーを返し、フェーズ2〜3はメインの会話で続ける。**【正解】**
- B) メインの会話ですべてのフェーズを行い、ファイルを通過しながらコンテキスト使用量を削減するために定期的に`/compact`を使用する。
- C) `--continue`付きのヘッドレスモードに切り替え、継続性を維持するために明示的なコンテキストサマリーをバッチ呼び出し間で渡す。
- D) CLAUDE.mdでエラーハンドリングパターンを定義し、共有メモリファイルに一貫性を依存しながら複数のセッションにわたってバッチでファイルを処理する。

**Aの理由：** 探索サブエージェントは詳細な発見出力を別のコンテキストに隔離し、メインの会話には簡潔なサマリーのみを返します。これにより、保持されたコンテキストが最も価値あるフェーズ2とフェーズ3（共同設計と一貫した実装）のためにメインのコンテキストウィンドウを保存します。

---

## シナリオ：カスタマーサポートエージェント

---

## 問題46（シナリオ：カスタマーサポートエージェント）

**状況：** テスト中、エージェントが注文ステータスについて尋ねるユーザーに対して`lookup_order`がより適切であるにもかかわらず、よく`get_customer`を呼び出すことに気づきます。この問題に対処するために最初に確認すべきことは何ですか？

**最初に確認すべきことは何ですか？**

- A) 注文関連リクエストを検出して`lookup_order`に直接ルーティングする前処理分類器を実装する。
- B) 選択を簡素化するためにエージェントが利用できるツールの数を減らす。
- C) ツールの選択を改善するためにすべての可能な注文リクエストパターンをカバーするFew-shotサンプルをシステムプロンプトに追加する。
- D) ツールの説明が各ツールの目的を明確に区別しているか確認する。**【正解】**

**Dの理由：** ツールの説明はモデルが呼び出すツールを決定するための主要な入力です。エージェントが一貫して間違ったツールを選択する場合、最初の診断ステップはツールの説明が各ツールの目的と使用境界を明確に分離していることを確認することです。

---

## 問題47（シナリオ：カスタマーサポートエージェント）

**状況：** エージェントは94%の精度で単一の問題リクエスト（例：「注文#1234の返金が必要です」）を処理します。しかし、顧客が1つのメッセージに複数の問題を含めると（例：「注文#1234の返金が必要で、注文#5678の配送先住所も変更したいです」）、ツール選択の精度は58%に落ちます。エージェントは通常1つの問題のみを解決するか、リクエスト間でパラメータを混同します。マルチ問題リクエストの信頼性を最も効果的に改善するアプローチは何ですか？

**最も効果的なアプローチは何ですか？**

- A) 別のモデル呼び出しを使ってマルチ問題メッセージを別々のリクエストに分解し、それぞれを独立して処理し、結果をマージする前処理層を実装する。
- B) 関連するツールをより少ない汎用ツールに統合する。
- C) マルチ問題リクエストに対する正しい推論とツールシーケンスを示すFew-shotサンプルをプロンプトに追加する。**【正解】**
- D) エージェントが見落とした問題を自動的に再プロンプトして解決する応答バリデーションを実装する。

**Cの理由：** エージェントは既に単一の問題については高いパフォーマンスを発揮しています。必要なのは、複数の問題を分解してルーティングし、パラメータを分離するパターンのガイダンスです。Few-shotサンプルによってその判断ロジックを最も直接的に示すことができます。Aは追加インフラを必要とする過剰なエンジニアリングです。Bはツールの専門性を損なわせます。Dは根本原因ではなく症状を後追いするだけです。

---

## 問題48（シナリオ：カスタマーサポートエージェント）

**状況：** 本番ログによると、「注文#1234の返金」のようなシンプルなリクエストでは、エージェントは3〜4回のツール呼び出しで91%の成功率で解決します。しかし、「2回請求された、割引が適用されていない、キャンセルしたい」のような複雑なリクエストでは、平均12回以上のツール呼び出しで成功率わずか54%です。エージェントは問題を順次調査し、各問題ごとに冗長な顧客データを取得することが多いです。複雑なリクエストの処理を最も効果的に改善する変更は何ですか？

**最も効果的な変更は何ですか？**

- A) フェーズ間に明示的な確認チェックポイントを追加し、次の問題に移る前に各問題の解決後に進捗を記録するようエージェントに要求する。
- B) `get_customer`、`lookup_order`、請求関連ツールを単一の`investigate_issue`ツールに統合してツールの数を減らす。
- C) リクエストを別々の問題に分解し、共有顧客コンテキストを使いながら各問題を並列で調査し、最終解決策を統合する。**【正解】**

- D) 様々な多面的な請求シナリオに対する理想的なツール呼び出しシーケンスを示すFew-shotサンプルをシステムプロンプトに追加する。

**Cの理由：** 問題を分解して共有顧客コンテキストで並列調査することで、2つの主要な問題を同時に解決します。共有コンテキストを再利用することで冗長なデータ取得を排除し、1回の統合解決策を統合する前に調査を並列化することでツール呼び出しの合計ループ数を削減します。Aはシーケンシャルなボトルネックを維持します。Bはツールの専門性を失わせます。Dは根本的な並列化の問題を解決しません。

---

## 問題49（シナリオ：カスタマーサポートエージェント）

**状況：** エージェントの初回解決率は55%で、目標の80%を大きく下回っています。ログを見ると、写真証拠付きの破損品に対する標準交換のようなシンプルなケースをエスカレーションしている一方、ポリシー例外が必要な複雑な状況を自律的に処理しようとしています。エスカレーションのキャリブレーションを改善する最も効果的な方法は何ですか？

**最も効果的な方法は何ですか？**

- A) 各レスポンスの前に自己評価信頼度（1〜10）を求め、信頼度が閾値を下回ったら自動的に人間にルーティングする。
- B) メインエージェントが処理を開始する前にどのリクエストがエスカレーションを必要とするかを予測するよう、過去のチケットで訓練された別の分類器モデルをデプロイする。
- C) エスカレーションすべき状況と自律的に解決すべき状況を示すFew-shotサンプルとともに、明示的なエスカレーション基準をシステムプロンプトに追加する。**【正解】**
- D) 顧客のフラストレーションレベルを判断する感情分析を実装し、ネガティブな感情の閾値を超えたら自動的にエスカレーションする。

**Cの理由：** 明示的なエスカレーション基準とFew-shotサンプルは、根本原因—シンプルなケースと複雑なケースの間の不明確な判断境界—に直接対処します。追加インフラを必要とせず、エスカレーションすべきタイミングと自律的に解決すべきタイミングをエージェントに教える最も効果的で適切な最初の介入です。Aは信頼性がありません（モデルは自信を持って誤る可能性があります）。Bは過剰なエンジニアリングです。Dは別の問題（感情 ≠ ケースの複雑さ）を解決しようとしています。

---

## 問題 50（シナリオ: カスタマーサポートエージェント）

**状況:** `get_customer` と `lookup_order` を呼び出した後、エージェントは利用可能なすべてのシステムデータを持っていますが、依然として不確実性に直面しています。`escalate_to_human` を呼び出す最も正当な状況はどれですか？

**どの状況がエスカレーションに最も正当ですか？**

- A) 顧客が昨日出荷された明日届く注文をキャンセルしたい。顧客が商品を受け取った後に気が変わる可能性があるためエスカレーションすべきです。
- B) 顧客が注文を受け取っていないと主張しますが、追跡では 3 日前に住所で配達・署名済みであることが示されています。矛盾する証拠を提示すると顧客関係が損なわれる可能性があるためエスカレーションすべきです。

- C) 顧客が競合他社の価格マッチングを要求します。ポリシーは 14 日以内の自社サイトでの価格下落に対して価格調整を許可しますが、競合他社の価格については何も言及していません。ポリシーの解釈のためにエスカレーションすべきです。 **[正解]**
- D) 顧客のメッセージに請求の質問と商品の返品の両方が含まれています。人間が 1 回のインタラクションで両方の問題を調整できるようにエスカレーションすべきです。

**C の理由:** これは真のポリシーのギャップです: 会社のルールは自社サイトでの価格下落をカバーしていますが、競合他社の価格マッチングには対応していません。エージェントはポリシーを作り上げてはならず、既存のルールをどのように解釈または拡張するかについて人間の判断のためにエスカレーションする必要があります。

---

## 問題 51（シナリオ: カスタマーサポートエージェント）

**状況:** 本番ログでは、12% のケースでエージェントが `get_customer` をスキップして顧客が提供した名前のみを使用して `lookup_order` を直接呼び出し、時に誤って識別されたアカウントと誤った返金につながることが示されています。

**この信頼性の問題を最も効果的に修正する変更は何ですか？**

- A) エージェントが顧客から自発的に注文の詳細を提供している場合でも、常に最初に `get_customer` を呼び出すことを示す Few-shot 例を追加する。
- B) 各リクエストを分析してそのリクエストタイプに適切なツールのサブセットのみを有効化するルーティング分類器を実装する。
- C) `get_customer` が確認された顧客識別子を返すまで `lookup_order` と `process_refund` をブロックするプログラム的な前提条件を追加する。 **[正解]**
- D) `get_customer` による顧客確認がすべての注文操作の前に必須であることを述べるシステムプロンプトを強化する。

**C の理由:** プログラム的な前提条件は必要なシーケンシングが従われることを決定論的に保証します。LLM の動作に関係なく確認をスキップする可能性を排除するため最も効果的なアプローチです。

---

## 問題 52（シナリオ: カスタマーサポートエージェント）

**状況:** 本番メトリクスでは、複雑な請求紛争やマルチ注文の返品を解決する際、解決策が技術的に正しい場合でも顧客満足スコアがシンプルなケースより 15% 低いことが示されています。根本原因分析では、エージェントが正確な解決策を提供しているが根拠を一貫性なく説明していることが示されています: 時に関連するポリシーの詳細を省略し、時にタイムライン情報や次のステップを見逃します。人間の監督を追加せずに解決策の品質を改善したい。

**どのアプローチが最も効果的ですか？**

- A) エージェントが下書きレスポンスの完全性を評価する自己批評ステージを追加する——顧客の問題を解決し、関連するコンテキストを含め、フォローアップの質問を見越しているかを確認する。 **[正解]**
- B) エージェントが閉じる前に「これで完全に問題が解決されましたか？」と尋ねる確認ステージを追加し、顧客が必要に応じて追加情報を要求できるようにする。

- C) 複雑なケースには Haiku から Sonnet にモデルをアップグレードし、定義された複雑さメトリクスに基づいてルーティングする。
- D) ポリシーコンテキスト、タイムライン、次のステップを含める方法を示す 5 つの一般的な複雑なケースタイプの完全な説明を示す Few-shot 例をシステムプロンプトに実装する。

**A の理由:** 自己批評ステージ（評価者-最適化者パターン）は、エージェントが提示する前に具体的な基準——ポリシーコンテキスト、タイムライン、次のステップなど——に対して自分の下書きを評価するよう強制することで、一貫性のない説明の完全性に直接対処します。人間の監督なしにケース固有のギャップをキャッチします。

---

## 問題 53（シナリオ: カスタマーサポートエージェント）

**状況:** 本番メトリクスでは、エージェントが解決ごとに平均 4 以上の API ループをかけていることが示されています。分析では、Claude が両方が最初に必要な場合でも `get_customer` と `lookup_order` を別々の逐次ターンでリクエストすることが多いことが明らかになっています。

**ループ数を削減する最も効果的な方法は何ですか？**

- A) リクエストされたツールと並行して必要になる可能性の高いツールを自動的に呼び出し、リクエストされたものに関係なくすべての結果を返す推測的実行を実装する。
- B) Claude がより自然にツールリクエストを計画して組み合わせるためにより多くの余地を与えるために `max_tokens` を増加する。
- C) 一般的なルックアップの組み合わせを単一の呼び出しにまとめた `get_customer_with_orders` などの複合ツールを作成する。
- D) 関連するツールリクエストを 1 つのターンにまとめて次の API 呼び出しの前にすべての結果をまとめて返すようにプロンプトで Claude に指示する。 **[正解]**

**D の理由:** 関連するツールリクエストを単一のターンにまとめるよう Claude にプロンプトすることで、複数のツールを一度にリクエストするというその本来の能力を活用します。最小限のアーキテクチャ変更で逐次呼び出しパターンを直接修正します。

---

## 問題 54（シナリオ: カスタマーサポートエージェント）

**状況:** 本番ログでは一貫したパターンが示されています: 顧客が特定の金額（例: 「私が言及した 15% の割引」）を参照するが、エージェントが誤った値で応答します。調査では、これらの詳細が 20 以上のターン前に言及され、「プロモーション価格について議論された」などの曖昧なサマリーに凝縮されていることが示されています。

**どの修正が最も効果的ですか？**

- A) 要約のしきい値を 70% から 85% に増加させて、要約がトリガーされる前に会話により多くの余地を与える。
- B) 外部ストレージに完全な会話履歴を保存し、エージェントが「私が言及したように」などの参照を検出した場合に取得を実装する。
- C) トランザクションの事実（金額、日付、注文番号）を要約された履歴の外の永続的な「ケース事実」ブロックに抽出し、すべてのプロンプトに含める。 **[正解]**

- D) すべての数値、パーセンテージ、日付、顧客が述べた期待を逐語的に保存するように要約プロンプトを修正する。

**C の理由:** 要約は本質的に正確な詳細を失います。トランザクションの事実を要約された履歴の外の構造化された「ケース事実」ブロックに抽出することで、何ターン要約されていても重要な情報がすべてのプロンプトで確実に利用できるようになります。

---

## 問題 55（シナリオ: カスタマーサポートエージェント）

**状況:** `get_customer` ツールは名前で検索する際にすべての一致を返します。現在、複数の結果がある場合、Claude は最も最近の注文を持つ顧客を選択しますが、本番データでは曖昧な一致の 15% で誤ったアカウントを選択することが示されています。

**どのように対処すべきですか？**

- A) 85% の信頼度以上で自律的に行動し、しきい値を下回ると明確化を要求する信頼スコアリングシステムを実装する。
- B) `get_customer` が複数の一致を返す場合、顧客固有のアクションを実行する前に追加の識別子（メール、電話番号、または注文番号）を要求するよう Claude に指示する。 **[正解]**
- C) ランキングアルゴリズムに基づいて最も可能性の高い単一の一致のみを返すよう `get_customer` を変更し、曖昧さを排除する。
- D) 曖昧な一致に対する正しい推論とツールシーケンシングを示す Few-shot 例をプロンプトに追加する。

**B の理由:** 追加の識別子をユーザーに要求することは、曖昧さを解決する最も信頼性の高い方法です。なぜなら、ユーザーは自分のアイデンティティについて決定的な知識を持っているからです。1 回の余分な会話ターンは、誤ったアカウントを選択する 15% のエラー率を排除するためのわずかなコストです。

---

## 問題 56（シナリオ: カスタマーサポートエージェント）

**状況:** 本番ログでは一貫したパターンが示されています: 顧客がメッセージに「アカウント」という単語を含める場合（例: 「昨日した注文についてアカウントを確認したい」）、エージェントは 78% の確率で最初に `get_customer` を呼び出します。顧客が「アカウント」なしで類似のリクエストを表現する場合（例: 「昨日した注文を確認したい」）、93% の確率で最初に `lookup_order` を呼び出します。ツールの説明は明確で曖昧さがありません。

**この不一致の最も可能性の高い根本原因は何ですか？**

- A) システムプロンプトには「アカウント」などの用語に基づいて動作を誘導するキーワード敏感な命令が含まれており、意図しないツール選択パターンを作り出している。 **[正解]**
- B) モデルの基礎トレーニングは「アカウント」の用語と顧客関連の操作との間に連想を作り出し、ツールの説明を上書きする。
- C) モデルはマルチコンセプトメッセージにより多くのトレーニングデータが必要で、アカウントと注文の両方の用語を含む例でファインチューニングすべきです。
- D) ツールの説明には各ツールを使用すべきでない場合を指定する追加のネガティブ例が必要です。

**A の理由:** 系統的なキーワード駆動のパターン（78% vs 93%）は、システムプロンプトの明示的なルーティングロジックが「アカウント」という単語に反応して顧客関連ツールへとエージェントを誘導していることを強く示しています。ツールの説明がすでに明確なため、不一致はプロンプトレベルの命令が意図しない動作の誘導を作り出していることを指し示しています。

---

## 問題 57（シナリオ: カスタマーサポートエージェント）

**状況:** 本番ログでは、「注文 #12345 を確認してください」のようなリクエストに対してエージェントが `lookup_order` を呼び出す代わりに `get_customer` を呼び出すことが多いことが示されています。両方のツールの説明は最小限（「顧客情報を取得します」/「注文の詳細を取得します」）で、類似した見た目の識別子フォーマットを受け付けます。

**ツール選択の信頼性を改善する最初のステップとして最も効果的なものは何ですか？**

- A) 各ターンの前にユーザー入力を分析し、検出されたキーワードと ID パターンに基づいて正しいツールを事前選択するルーティング層を実装する。
- B) 両方のツールを単一の `lookup_entity` に統合し、任意の識別子を受け付けて内部でどのバックエンドをクエリするかを決定する。
- C) 注文関連のクエリを `lookup_order` にルーティングする 5〜8 の例を含む、正しいツール選択パターンを示す Few-shot 例をシステムプロンプトに追加する。
- D) 各ツールの説明を入力フォーマット、サンプルクエリ、エッジケース、類似ツールと比較してどのような場合に使用するかの境界で拡張する。 **[正解]**

**D の理由:** 入力フォーマット、サンプルクエリ、エッジケース、明確な境界でツールの説明を拡張することで、根本原因——LLM に類似ツールを区別するための十分な情報を与えない最小限の説明——を直接修正します。LLM がツール選択のために使用する主要なメカニズムを改善する低コストで高いインパクトの最初のステップです。

---

## 問題 58（シナリオ: カスタマーサポートエージェント）

**状況:** サポートエージェントのエージェントループを実装しています。各 Claude API 呼び出しの後、ループを続行するか（リクエストされたツールを実行して Claude を再度呼び出す）または停止するか（最終回答を顧客に提示する）を決定する必要があります。

**この決定を何が決定しますか？**

- A) Claude のレスポンスの `stop_reason` フィールドを確認する—— `tool_use` の場合は続行し、`end_turn` の場合は停止する。 **[正解]**
- B) 「終わりました」や「他に何かお手伝いできることはありますか？」などのフレーズを Claude のテキストで解析する——自然言語シグナルがタスクの完了を示す。
- C) 最大反復回数（例: 10 回の呼び出し）を設定し、Claude がさらに作業が必要かどうかを示しているかどうかに関係なく到達時に停止する。
- D) レスポンスにアシスタントのテキストコンテンツが含まれているかどうかを確認する——Claude が説明テキストを生成した場合、ループを終了すべきです。

**A の理由:** `stop_reason` はループ制御のための Claude の明示的な構造化シグナルです: `tool_use` はツールを実行して結果を返してほしいことを示し、`end_turn` は Claude がレスポンスを完了してループが終了すべきことを示します。

---

## 問題 59（シナリオ: カスタマーサポートエージェント）

**状況:** 本番ログでは、エージェントが MCP ツールからの出力を誤って解釈していることが示されています: `get_customer` からの Unix タイムスタンプ、`lookup_order` からの ISO 8601 日付、数値のステータスコード（1=保留中、2=出荷済み）。一部のツールは変更できないサードパーティの MCP サーバーです。

**データフォーマットの正規化にどのアプローチが最も保守しやすいですか？**

- A) ツール出力をインターセプトしてエージェントが処理する前にフォーマット変換を適用するための PostToolUse フックを使用する。 **[正解]**
- B) 制御できるツールを変更して人間が読めるフォーマットを返すようにし、サードパーティツールのラッパーを作成する。
- C) エージェントがすべてのデータ取得後に呼び出して値を変換する `normalize_data` ツールを作成する。
- D) 各ツールのデータ規約を説明する詳細なフォーマットドキュメントをシステムプロンプトに追加する。

**A の理由:** PostToolUse フックはすべてのツール出力（サードパーティ MCP サーバーのデータを含む）をエージェントが処理する前にインターセプトして正規化するための集中した決定論的なポイントを提供します。変換がコードに存在して均一に適用されるため LLM の解釈に頼るよりも保守しやすいです。

---

## 問題 60（シナリオ: カスタマーサポートエージェント）

**状況:** 本番ログでは、エージェントが「最近の購入について助けが必要です」のような曖昧なクエリに対して特に `lookup_order` の方が適切なのに `get_customer` を選択することがあることが示されています。ツール選択を改善するためにシステムプロンプトに Few-shot 例を追加することにしました。

**どのアプローチが最も効果的ですか？**

- A) 曖昧なケースをカバーする各ツールの説明に明示的な「使用する場合」と「使用しない場合」のガイダンスを追加する。
- B) ツール別にグループ化した例を追加する——すべての `get_customer` シナリオをまとめ、次にすべての `lookup_order` シナリオをまとめる。
- C) それぞれが考えられる代替案よりもなぜ一方のツールが選ばれたかの根拠を持つ、曖昧なシナリオをターゲットにした 4〜6 の例を追加する。 **[正解]**
- D) 各ツールの典型的なシナリオの正しいツール選択を示す、明確で曖昧さのないリクエストの 10〜15 の例を追加する。

**C の理由:** エラーが発生する具体的な曖昧なシナリオをターゲットにした Few-shot 例を、代替案よりも一方のツールを好む理由の明示的な根拠と共に提供することで、エッジケースに必要な比較的な意思決定

プロセスをモデルに教えます。これは汎用的な例や宣言的なルールよりも効果的です。

---

## 問題 61（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** `remove_team_member` ツールは、実行前に影響をプレビューするための `dry_run: boolean` パラメーターを使用しています。本番監視では、エージェントが `dry_run=false` で直接呼び出すことでプレビューステップをバイパスしていることが示されています。各削除操作の前に、ユーザーが明示的に確認するプレビューが行われるように保証する必要があります。

**最も信頼性の高いアプローチはどれですか？**

- A) 同一パラメーターでの `dry_run=true` 呼び出しが過去 60 秒以内に発生した場合のみ `dry_run=false` を許可するサーバー側検証を追加する。
- B) ツールを確認が必要なものとして注釈し、注釈付きツールへの呼び出しを転送する前にユーザーの承認を求めるようにオーケストレーション層を設定する。
- C) ツールの説明に詳細な指示と Few-shot 例を追加し、エージェントが常に `dry_run=true` で最初に呼び出し、再度呼び出す前にユーザーの確認を待つよう要求する。
- D) 2つのツールに置き換える：`preview_remove_member` は影響の詳細と一回限りの確認トークンを返し、`execute_remove_member` はそのトークンを要求し、実行をプレビューに結びつける。**【正解】**

**D の理由：** トークンバインディングアプローチにより、事前プレビューなしでの実行がアーキテクチャ上不可能になります—実行ツールは文字通り、プレビューツールだけが生成できるトークンを必要とします。これは、LLMによる指示の遵守 (C)、タイミングヒューリスティクス (A)、またはオーケストレーションインフラ (B) に依存するのではなく、コードレベルで制約を強制する唯一のアプローチです。

---

## 問題 62（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** 本番監視では、`search_catalog` ツールが 12% の時間失敗することが示されています：8% は再試行で成功するネットワークタイムアウト、4% は再試行に関わらず決して成功しないクエリ構文エラーです。現在、両方のエラータイプは同一に返されており、無駄な再試行が発生しています。

**ツールのエラー処理をどのように修正すべきですか？**

- A) ネットワークエラーと構文エラーを区別する方法を示す Few-shot 例をシステムプロンプトに追加する。
- B) すべてのエラーに均一に指数バックオフ再試行ロジックを適用する。
- C) ツール内でネットワークタイムアウトに対して自動再試行とバックオフを実装し、構文エラーはパラメーター検証の詳細を含めて即座に返す。**【正解】**
- D) すべてのエラーを `retryable` ブールフラグとエラータイプの詳細付きで返す。

**C の理由：** 一時的なエラーに対してツールレベルで再試行を処理することが正しい抽象境界です—ツールはエラータイプについて確実な知識を持ち、エージェントがフラグ (D) を解釈したりプロンプト指示 (A) に従ったりすることに依存せずに、確定的な再試行ロジックを実装できます。均一なバックオフ (B) は決して成功しない構文エラーに時間を無駄にします。

---

## 問題 63（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** 投資戦略について複数のターンで議論する中で、ユーザーは「リスク許容度が非常に低い」と述べ、その後「リターンを最大化したい」と述べました。今、「何に投資すべきか？」と尋ねます。

**ユーザーの実際の優先事項に沿った推奨を最も確実に保証するアプローチはどれですか？**

- A) 矛盾を表面化し、どちらがより重要かをユーザーに明確にするよう求める。**【正解】**
- B) 両方のシナリオに対して別々の推奨を提供する。
- C) 最近述べた好みに基づいて進める。
- D) 矛盾に対処せずにバランスのとれたポートフォリオを推奨する。

**A の理由：** ユーザーの好みが直接矛盾する場合、矛盾を表面化して明確化を求めることが、推奨がユーザーの真の意図に沿っていることを保証する唯一の方法です。リターンの最大化と低いリスク許容度は根本的に相容れない目標であり、人間の判断が必要です。

---

## 問題 64（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** ユーザーは複数の会話ターンにわたってプレイリストの好みを洗練させます。ユーザーが「ジャズが好き」と言った2つのメッセージ後、Claudeは「どのジャンルが好きですか？」と尋ねます。

**最も可能性の高い原因は何ですか？**

- A) Claudeは会話記憶を維持するためにベクターデータベース接続が必要です。
- B) モデルのコンテキストウィンドウが超過されました。
- C) Claude API には `session_id` パラメーターが必要です。
- D) アプリケーションが `messages` 配列に以前のメッセージを含めていません。**【正解】**

**D の理由：** Claudeはサーバー側の記憶を持ちません—各 API 呼び出しはステートレスです。各リクエストの `messages` 配列に完全な会話履歴を含めないと、Claudeは以前のターンで何が言われたかを知る方法がありません。ベクターデータベース (A) と `session_id` (C) は Claude アーキテクチャの一部ではなく、2つのメッセージ交換でのコンテキストウィンドウオーバーフロー (B) は不可能です。

---

## 問題 65（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** 40 分間の料理セッション後、会話は 78,000 トークンに達しました。履歴にはアレルギー、レシピのスケーリング、料理用語の説明、一般的な議論が含まれています。重要な情報を保持しながらトークンを削減する必要があります。

**保持とトークン削減のバランスが最も優れているアプローチはどれですか？**

- A) 会話履歴全体を要約する。
- B) 最新の 20,000 トークンのみを保持する。
- C) 重要な構造化データ（アレルギー、数量、好み）を抽出し、一般的な議論を要約し、最近の交換を逐語的に保持する。**【正解】**
- D) 完全な会話を外部に保存し、セマンティック検索を通じて関連部分を取得する。

**C の理由：** ハイブリッドアプローチは最低コストで最高価値の情報を保持します。アレルギーやレシピの数量などの重要な事実はコンパクトな構造化ブロックに抽出され（要約中に発生する精度損失を防止）、一般的な議論は要約され、最近の交換は会話の一貫性のために逐語的に保持されます。オプション A と B は重要な食事情報を失うリスクがあり、D は単一の料理セッションには過剰です。

---

## 問題 66（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** ユーザーは、長い会話の中でアシスタントが以前のトピックや好みを追跡できなくなると報告しています。現在の実装では、最後の 25 メッセージペアのみを保持しています。

**最も効果的な解決策は何ですか？**

- A) ハイブリッドアプローチ：古いメッセージを要約しながら、最近のメッセージを逐語的に保持する。**【正解】**
- B) 完全な会話履歴に対するベクター類似性検索。
- C) ウィンドウを 50 メッセージペアに増やす。
- D) 毎ターン削除されたメッセージを要約し、実行中の要約を先頭に追加する。

**A の理由：** ハイブリッドアプローチは問題の両方の側面に対処します：正確な最近のコンテキストを保持（会話の一貫性に不可欠）しながら、以前の好みの圧縮表現を維持します。ウィンドウを増やすこと (C) は同じ問題を先延ばしにするだけです。ベクター検索 (B) は現在のクエリとセマンティックに類似していない重要なコンテキストを見逃す可能性があります。

---

## 問題 67（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** ユーザーは、会話が 50 ターンを超えると遅延が増加し、コストが上昇すると報告しています。

**主な原因は何ですか？**

- A) 各 API リクエストに会話履歴全体が含まれます。**【正解】**
- B) モデルが徐々に長い応答を生成します。
- C) 履歴が増えるにつれてデータベース操作が遅くなります。
- D) モデルがより多くの処理を必要とする内部ユーザープロファイルを構築します。

**A の理由：** Claude の API は完全にステートレスです—各リクエストは `messages` 配列に完全な会話履歴を含める必要があります。会話が増えると、各リクエストがより多くのトークンを運び、処理遅延とコストの両方が直接増加します。モデルは呼び出し間で内部状態を維持しません (D は偽)。

---

## 問題 68（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** 3 ヶ月間の週次セッション後、会話履歴は 85,000 トークンに増加しました。ユーザーが「孤立のテーマについて何を結論付けましたか？」と尋ねると、アシスタントは以前の議論を参照する代わりに一般的な回答をします。

**最も効果的なアプローチは何ですか？**

- A) ローリングウィンドウ切り捨て。
- B) 主要な結論をキャプチャする段階的な要約。
- C) 関連する交換を取得するセマンティックエンベディング。**【正解】**
- D) 議論の結論をマークする構造化 XML タグを追加する。

**C の理由：** 会話履歴に対するセマンティック検索は、3 ヶ月分の議論にスケールしながら、要求に応じて特定の関連する交換を表面化できる唯一のアプローチです。ローリングウィンドウ切り捨て (A) は履歴の大部分を廃棄します。段階的な要約 (B) は議論を抽象化に圧縮し、ユーザーが尋ねる具体的な結論が失われます。

---

## 問題 69（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** QA テスト中、Claudeは最初の 10〜15 ターンはシステムプロンプトのガイドラインに従いますが、その後の応答は逸脱します。会話はまだトークン制限内です。

**最善の解決策は何ですか？**

- A) 行動ガイドラインを最初のユーザーメッセージに移動する。
- B) 20 ターン後に新しい会話を開始する。
- C) 会話のブレークポイントでガイドラインを強化するユーザーロールメッセージを挿入する。**【正解】**
- D) 事後検証を使用して非準拠の応答を再生成する。

**C の理由：** 行動リマインダーの定期的な注入は、履歴が蓄積するにつれて定期的な間隔で制約を再確立することで、指示のドリフトに直接対抗します。ガイドラインを最初のユーザーメッセージに移動すること (A) はその権威を低下させます。事後検証 (D) は予防的ではなく修正的であり、大幅な遅延を追加します。

---

## 問題 70（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** AI チューターには、教授方法論と適応ルールを定義する 2,800 トークンのシステムプロンプトがあります。12 ターン後、アシスタントは習熟度レベルを無視し始めます。

**最も効果的な修正は何ですか？**

- A) 4〜5 ターンごとにリマインダーを注入する。
- B) 冗長なルールを習熟度レベルの適応を示す Few-shot 例に置き換える。**【正解】**
- C) 重要なルールをシステムプロンプトの最後に配置する。
- D) 応答を評価し、難易度レベルが一致しない場合は再生成する。

**B の理由：** 宣言的なルールを含む 2,800 トークンのシステムプロンプトはドリフトに脆弱です。なぜなら、抽象的なルールはモデルに各ターンで推論を要求するからです。冗長なルールを正しい習熟度レベルの適応を示す具体的な Few-shot 例に置き換えることで、モデルにマッチングする明確な行動パターンが提供されます—これは多くのターンにわたって抽象的な指示よりも信頼性高く遵守されます。

---

## 問題 71（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** アシスタントは熱心なトーンを維持し、推論を説明し、明確化の質問をする必要があります。これらの行動ガイドラインはどこで定義すべきですか？

**これらの行動ガイドラインはどこで定義すべきですか？**

- A) 各ユーザーメッセージの前に追加する。
- B) システムプロンプトで定義する。**【正解】**
- C) 最初のアシスタントメッセージで定義する。
- D) 環境変数で定義する。

**B の理由：** システムプロンプトは、会話全体を通じて適用される永続的な行動制約とガイドラインのために特別に設計されています。各ユーザーメッセージの前に追加すること (A) は冗長なオーバーヘッドです。最初のアシスタントメッセージ (C) は信頼性が低く、モデルは自身の以前の発言から逸脱する可能性があります。環境変数 (D) はモデルの動作に影響を与えません。

---

## 問題 72（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** ユーザーは「もちろん！」や「喜んでお手伝いします！」などの繰り返しの応答の書き出しを報告しています。

**最も効果的なアプローチは何ですか？**

- A) 直接的な応答の書き出しで部分的なアシスタントメッセージを追加する。**【正解】**
- B) 温度設定を下げる。
- C) 応答を後処理して挨拶を削除する。
- D) それらのフレーズを避けるためのシステムプロンプトの指示を追加する。

**A の理由：** 直接的な回答の書き出しでアシスタントの応答をプリフィルすることで、生成レベルで挨拶パターンを防止します—モデルは新しい書き出しフレーズを生成するのではなく、プリフィルから続けます。システムプロンプトの指示 (D) は役立つかもしれませんが、信頼性が低くなります。後処理 (C) は脆弱な回避策です。温度 (B) はランダム性を制御しますが、特定のフレーズパターンは制御しません。

---

## 問題 73（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** ユーザーがチャット中に、webhookがシステムにユーザーの荷物が発送されたことを通知します。アシスタントがこれを次の応答に自然に組み込むようにしたいです。

**最善のアプローチは何ですか？**

- A) 配送ステータスをシステムプロンプトに追加する。
- B) 即座の合成ユーザーメッセージを送信する。
- C) 毎ターンアシスタントにステータスツールを呼び出すよう強制する。
- D) ステータス更新を次のユーザーメッセージへのプレフィックスとして追加する。**【正解】**

**D の理由：** ステータス更新を次のユーザーメッセージへのプレフィックスとして追加することで、フローを中断することなく自然な会話の境界でリアルタイムのコンテキストを注入します。システムプロンプトの変更 (A) はセッションの再構築が必要です。合成ユーザーメッセージ (B) は自然な会話の流れを乱す可能性があります。毎ターンツール呼び出しを強制すること (C) はイベントが稀な場合に無駄です。

---

## 問題 74（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** ユーザーは「パーティーのために会場を予約して」などのリクエストを頻繁に送信します。アシスタントは 4 つ以上の明確化の質問をし、35% の放棄率を引き起こしています。

**トレードオフを最もよく改善するアプローチは何ですか？**

- A) 隠れたデフォルトで進める。
- B) すべての明確化の質問を一つの複合メッセージで尋ねる。
- C) 仮定を明示的に述べて、修正を求めながら進める。**【正解】**
- D) 構造化された受付フォームを使用する。

**C の理由：** 仮定を明示的に述べて進めることで、ユーザーに即座に有用な応答を提供しながら、誤った仮定を修正する能力を保持します。隠れたデフォルト (A) はユーザーに何が仮定されたかを伝えません。複合質問リスト (B) は依然としてユーザーからの初期努力を要求します。構造化フォーム (D) は摩擦を減らすのではなく増やします。

---

## 問題 75（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** アシスタントはコントラクターペルソナのシステムプロンプトを使用しています。初期のターンはルールに従いますが、ターン 7 までにアシスタントは一般的なアドバイスを提供します。会話の長さはわずか 2,500 トークンです。

**最も可能性の高い原因は何ですか？**

- A) システムプロンプトは初期動作のみを確立します。
- B) ターンが蓄積するにつれてモデルの注意が弱まります。
- C) 蓄積されたアシスタントの応答がシステムプロンプトの影響を希釈します。**【正解】**
- D) システムプロンプトは一度だけ送信されます。

**C の理由：** 会話履歴にアシスタントの応答が蓄積するにつれて、システムプロンプトの行動制約を反映するテキストの割合が、増大するアシスタント生成コンテンツに対して減少します。モデルは短いトークン長でも、システムプロンプトの指示よりも自身の以前の出力パターンにますます従うようになり、ドリフトを複合させます。

---

## 問題 76（シナリオ: 会話型AIアーキテクチャパターン）

**状況：** ユーザーは「レポートを手伝ってもらえますか？」などの漠然としたリクエストをします。アシスタントは複数の質問（どのレポート？どんな手伝い？締め切りは？）で応答し、40% の放棄率を引き起

こしています。

**最善の解決策は何ですか？**

- A) 合理的な仮定を行い、明示的に述べ、調整を提供する。**【正解】**
- B) 応答する前に小さなモデルで曖昧さを分類する。
- C) 仮定を述べずに事前定義された解釈を使用する。
- D) アシスタントを毎ターン 1 つの明確化の質問に制限する。

**A の理由：** 合理的な明示された仮定で進めることで、ユーザーに情報提供しコントロールを保持させながら、往復のやり取りを完全に排除します。静かな事前定義の解釈 (C) は応答がユーザーの意図に合わない場合にユーザーを混乱させます。1 質問制限 (D) は依然として往復のターンを必要とします。小さな分類モデル (B) はコア UX 問題を解決せずに遅延とインフラの複雑さを追加します。

---

# 実践演習

## 演習 1: エスカレーションロジックを持つマルチツールエージェント

**目標:** ツール統合、構造化エラーハンドリング、エスカレーションを持つエージェントループを設計する。

**ステップ:**

1. 詳細な説明を持つ 3〜4 の MCP ツールを定義する（ツール選択をテストするために 2 つの類似ツールを含める）
2. `stop_reason`（`"tool_use"` / `"end_turn"`）を確認するエージェントループを実装する
3. 構造化エラーレスポンスを追加する: `errorCategory`、`isRetryable`、説明
4. しきい値を超える操作をブロックしてエスカレーションにルーティングするインターセプターフックを実装する
5. マルチアスペクトのリクエストでテストする

**ドメイン:** 1（エージェントアーキテクチャ）、2（ツールと MCP）、5（コンテキストと信頼性）

---

## 演習 2: チーム開発のための Claude Code の設定

**目標:** CLAUDE.md、カスタムコマンド、パス固有のルール、MCP サーバーを設定する。

**ステップ:**

1. ユニバーサル標準を持つプロジェクトレベルの CLAUDE.md を作成する
2. 異なるコード領域のための YAML フロントマターを持つ `.claude/rules/` ファイルを作成する（`paths: ["src/api/**/*"]`、`paths: ["**/*.test.*"]`）
3. `context: fork` と `allowed-tools` を持つ `.claude/skills/` のプロジェクトスキルを作成する
4. 環境変数付きの `.mcp.json` に MCP サーバーを設定し、`~/.claude.json` に個人のオーバーライドを設定する

5. 異なる複雑さのタスクでプランニングモード vs 直接実行をテストする

**ドメイン:** 3（Claude Code 設定）、2（ツールと MCP）

---

## 演習 3: 構造化データ抽出パイプライン

**目標:** JSON スキーマ、構造化出力のための `tool_use`、検証/リトライループ、バッチ処理。

**ステップ:**

1. JSON スキーマを持つ抽出ツールを定義する（必須/任意フィールド、"other" を含む enum、null 許容フィールド）
2. 検証ループを構築する: エラー時に、ドキュメント、誤った抽出、特定の検証エラーを使ってリトライする
3. 異なる構造のドキュメントのための Few-shot 例を追加する
4. Message Batches API 経由でバッチ処理を使用する: 100 ドキュメント、`custom_id` で失敗を処理する
5. 人間へのルーティング: フィールドレベルの信頼スコア、ドキュメントタイプ分析

**ドメイン:** 4（プロンプトエンジニアリング）、5（コンテキストと信頼性）

---

## 演習 4: マルチエージェント調査パイプラインの設計とデバッグ

**目標:** サブエージェントのオーケストレーション、コンテキスト受け渡し、エラー伝播、ソーストラッキングを持つ合成。

**ステップ:**

1. 2 以上のサブエージェントを持つコーディネーター（`allowedTools` に `"Task"` を含み、プロンプトにコンテキストを明示的に渡す）
2. 1 回のレスポンスで複数の `Task` 呼び出しによりサブエージェントを並列実行する
3. 構造化されたサブエージェント出力を要求する: 主張、引用、ソース URL、公開日
4. サブエージェントのタイムアウトをシミュレートする: 構造化エラーコンテキストをコーディネーターに返して部分的な結果で続行する
5. 競合するデータでテストする: 帰属付きで両方の値を保存; 確認済みと争われた発見事項を分離する

**ドメイン:** 1（エージェントアーキテクチャ）、2（ツールと MCP）、5（コンテキストと信頼性）

---

# 付録: 技術とコンセプト

| 技術 | 主要な側面 |
|---|---|
| **Claude Agent SDK** | AgentDefinition、エージェントループ、`stop_reason`、フック（PostToolUse）、Task によるサブエージェントの生成、`allowedTools` |

| | |
|---|---|
| **Model Context Protocol (MCP)** | MCP サーバー、ツール、リソース、`isError`、ツールの説明、`.mcp.json`、環境変数 |
| **Claude Code** | CLAUDE.md 階層、glob パターンを持つ `.claude/rules/`、`.claude/commands/`、SKILL.md を持つ `.claude/skills/`、プランニングモード、`/compact`、`--resume`、`fork_session` |
| **Claude Code CLI** | 非インタラクティブモードのための `-p` / `--print`、`--output-format json`、`--json-schema` |
| **Claude API** | JSON スキーマを持つ `tool_use`、`tool_choice`（"auto"/"any"/強制）、`stop_reason`、`max_tokens`、システムプロンプト |
| **Message Batches API** | 50% の節約、最大 24 時間のウィンドウ、`custom_id`、マルチターンツール呼び出しなし |
| **JSON スキーマ** | 必須 vs 任意、null 許容フィールド、enum 型、"other" + 詳細、厳密モード |
| **Pydantic** | スキーマ検証、意味エラー、検証/リトライループ |
| **組み込みツール** | Read、Write、Edit、Bash、Grep、Glob — 目的と選択基準 |
| **Few-shot プロンプティング** | 曖昧な状況のためのターゲットを絞った例、新しいパターンへの一般化 |
| **プロンプトチェイニング** | 集中したパスへの逐次分解 |
| **コンテキストウィンドウ** | トークン予算、逐次要約、「中間で失われる」、スクラッチパッドファイル |
| **セッション管理** | 再開、`fork_session`、名前付きセッション、コンテキストの分離 |
| **信頼度キャリブレーション** | フィールドレベルスコアリング、ラベル付きセットでのキャリブレーション、層別サンプリング |

# 範囲外のトピック

以下の隣接トピックは**試験に出題されません**:

- Claude モデルのファインチューニングまたはカスタムモデルのトレーニング
- Claude API の認証、請求、またはアカウント管理
- 特定のプログラミング言語やフレームワークでの詳細な実装（ツール/スキーマ設定に必要な範囲を除く）
- MCP サーバーのデプロイまたはホスティング（インフラ、ネットワーキング、コンテナオーケストレーション）
- Claude の内部アーキテクチャ、トレーニングプロセス、またはモデルの重み
- Constitutional AI、RLHF、または安全トレーニングの方法論
- 埋め込みモデルまたはベクトルデータベースの実装の詳細

- コンピューター使用（ブラウザオートメーション、デスクトップインタラクション）
- 画像分析機能（ビジョン）
- ストリーミング API またはサーバー送信イベント
- レート制限、クォータ、または詳細な API コスト計算
- OAuth、API キーのローテーション、または認証プロトコルの詳細
- クラウドプロバイダー固有の設定（AWS、GCP、Azure）
- パフォーマンスベンチマークまたはモデル比較メトリクス
- プロンプトキャッシングの実装の詳細（存在することを知っている範囲を超えて）
- トークンカウントアルゴリズムまたはトークナイゼーションの詳細

---

# 準備の推奨事項

1. **Claude Agent SDK でエージェントを構築する** — ツール呼び出し、エラーハンドリング、セッション管理を含む完全なエージェントループを実装します。サブエージェントと明示的なコンテキスト受け渡しを練習します。
2. **実際のプロジェクトのために Claude Code を設定する** — CLAUDE.md 階層、`.claude/rules/` のパス固有のルール、`context: fork` と `allowed-tools` を持つスキル、MCP サーバー統合を使用します。
3. **MCP ツールを設計してテストする** — 類似のツールを区別する説明を書き、カテゴリとリトライフラグを持つ構造化エラーを返し、曖昧なユーザーリクエストに対してテストします。
4. **データ抽出パイプラインを構築する** — JSON スキーマを持つ `tool_use`、検証/リトライループ、任意/null 許容フィールド、Message Batches API 経由のバッチ処理を使用します。
5. **プロンプトエンジニアリングを練習する** — 曖昧なシナリオのための Few-shot 例、明示的なレビュー基準、大規模なコードレビューのためのマルチパスアーキテクチャを追加します。
6. **コンテキスト管理パターンを学習する** — 冗長な出力から事実を抽出し、スクラッチパッドファイルを使用し、コンテキストの制限を処理するためにサブエージェントに発見を委任します。
7. **エスカレーションとヒューマンインザループを理解する** — エスカレーションするタイミング（ポリシーのギャップ、明示的なユーザーリクエスト、進展不可能）と信頼度ベースのルーティングワークフロー。
8. **実際の試験前に模擬試験を受ける。** 同じシナリオと形式を使用します。