We offer free update service for one year!
https://www.certqueen.com

Exam : **CCDV-F**

Title : Claude Certified Developer - Foundations

Version : DEMO

1.Scenario: Developer Productivity with Claude You are building developer productivity tools using the Claude Agent SDK. The agent helps engineers explore unfamiliar codebases, understand legacy systems, generate boilerplate code, and automate repetitive tasks. It uses built-in tools (Read, Write, Bash, Grep, Glob) and integrates with MCP servers. During a legacy payment module analysis, the coordinator first assigns one subagent to map database tables and another to trace API callers. Both return useful findings. The coordinator then invokes a planning subagent to propose a migration strategy, but the plan ignores the database constraints and caller list already discovered, recommending changes that would break known integrations.
What should you change to make this orchestration more reliable?
A. Replace the specialized subagents with one long-running generalist subagent that performs discovery and planning together.
B. Instruct the planning subagent to infer missing dependencies from file names when prior findings are unavailable.
C. Allow subagents to message each other directly so the planning subagent can request missing analysis details.
D. Have the coordinator maintain shared investigation state and include relevant prior findings in each subsequent subagent prompt.
Answer: D
Explanation:
Coordinator-owned state is central to reliable multi-agent orchestration. In a coordinator-subagent pattern, the coordinator decomposes work, collects results, and decides which findings must be supplied to later subagents so they can produce grounded outputs.
The underlying principle is that subagents operate in *isolated task contexts*. They should not be designed as if they can automatically see prior coordinator conversations or other subagents' outputs. The coordinator should maintain a structured investigation state, such as discovered tables, caller lists, constraints, open questions, and confidence notes, then include the relevant subset in each downstream prompt.
Letting subagents communicate directly weakens the hub-and-spoke architecture by hiding information flow from the coordinator. Replacing specialists with one generalist can create context bloat and reduce the value of parallel specialized analysis. Asking a planning subagent to infer missing dependencies from file names is especially risky because it substitutes guesses for evidence.
For more on agent orchestration and subagent patterns, see the Agent SDK documentation and the Claude Code Sub-agents documentation.

2.Scenario: Structured Data Extraction You are building a structured data extraction system using Claude. The system extracts information from unstructured documents, validates output using JSON schemas, and maintains high accuracy. It must handle edge cases gracefully and integrate with downstream systems. Your extraction QA pass reviews Claude's JSON outputs before downstream ingestion. Reviewers dismiss many findings because the QA prompt flags harmless differences: inferred date formats, optional fields absent from the source, and wording variations that do not change extracted values. The current prompt says, "Check the extraction for accuracy and report any problems.".
What change would most effectively improve precision?
A. Increase the validation sample size and ask reviewers to manually ignore findings that are not actionable.

B. Add instructions that Claude should be conservative and report only findings where it feels highly confident.
C. Rewrite the QA prompt to define reportable errors, acceptable variations, and skip conditions with concrete examples for each category.
D. Require the QA pass to flag every schema field that is null, even when the source document omits it.
Answer: C
Explanation:
Explicit criteria are the most effective way to improve precision when a prompt produces false positives. A prompt like *"check for accuracy"* leaves Claude to infer what counts as an issue, so it may flag harmless formatting differences, valid nulls, or stylistic variations as problems.
The stronger approach is to define reportable categories, acceptable variations, and skip conditions. For example, report a value that contradicts the source or a source-present required field that was omitted, but skip optional fields absent from the document and normalized formats that preserve meaning.
Instructions such as *"be conservative"* or *"only report high-confidence findings"* are weak substitutes because they do not create operational decision boundaries. Flagging all null fields is also counterproductive in extraction systems, since nullable fields are often necessary to avoid fabrication when source data is absent.
The underlying principle is that precision improves when the model receives concrete decision rules and examples rather than broad quality goals. Learn more about prompt specificity and examples in Prompt Engineering.

3.Scenario: Structured Data Extraction You are building a structured data extraction system using Claude. The system extracts information from unstructured documents, validates output using JSON schemas, and maintains high accuracy. It must handle edge cases gracefully and integrate with downstream systems. Your invoice extractor currently asks Claude to "return valid JSON" with fields for vendor, invoice_date, line_items, and total_amount. In production, a small but persistent share of responses include markdown fences, explanatory text, or malformed commas, causing downstream parsers to reject otherwise correct extractions.
What change would most effectively improve structured output reliability?
A. Keep requesting JSON only, then strip markdown fences and trailing prose with regular expressions before parsing.
B. Add stronger prompt wording that forbids explanations, examples, markdown, comments, and any non-JSON characters in every response.
C. Define an extraction tool with a JSON schema for target fields, then consume the structured tool_use input directly.
D. Use assistant prefill with an opening brace and stop sequences, then parse the generated text as JSON.
Answer: C
Explanation:
Structured output reliability is highest when the application uses Claude's tool use mechanism with a JSON schema representing the desired extraction fields. The model emits a tool_use block whose input conforms to the declared schema, so the application consumes structured arguments rather than scraping JSON from natural language text.

The underlying principle is to avoid treating machine-readable output as ordinary prose. Prompt-only instructions, assistant prefill, and stop sequences can improve formatting, but they still rely on text generation and parsing. For production extraction pipelines, tool use with schemas directly addresses JSON syntax failures such as markdown fences, extra commentary, and malformed delimiters.
Regex cleanup is especially brittle because it creates an ad hoc repair layer that can silently corrupt data or fail on new formatting variants. Stronger wording is also insufficient when downstream systems require consistent parseability rather than best-effort compliance. Learn more in the Tool Use documentation and the Claude API & SDK documentation.

4.Scenario: Multi-Agent Research System You are building a multi-agent research system using the Claude Agent SDK. A coordinator agent delegates to specialized subagents: one searches the web, one analyzes documents, one synthesizes findings, and one generates reports. The system researches topics and produces comprehensive, cited reports. A user requests a report comparing how proposed AI copyright rules affect music licensing, model training data, and independent film production across the same jurisdictions and dates. Logs show the coordinator routes the entire request to one document analysis pass, then asks the report agent to write separate sections. The final report deeply covers licensing, barely addresses training data, and gives recommendations that conflict across sectors.
What workflow change would most effectively improve the result?
A. Split the request into distinct concern threads, investigate them in parallel with shared constraints, then synthesize one unified cross-sector report.
B. Process the concerns sequentially from highest commercial value to lowest, finalizing each report section before starting the next concern.
C. Add an instruction that the report agent should mention every requested sector at least once before delivering the final answer.
D. Ask every subagent to independently analyze the full user request, then concatenate their outputs into the final report document.
Answer: A
Explanation:
Correct workflow pattern: Compound requests should be decomposed into distinct concerns that can be investigated with focused attention, while shared constraints such as jurisdictions, dates, source requirements, and definitions are passed into each investigation. The coordinator can then synthesize the findings into a single report that reconciles interactions, conflicts, and dependencies across the concerns.
The underlying architectural principle is to separate *focused parallel investigation* from *unified synthesis*. A single broad pass risks attention dilution, while independent full-scope analyses create duplication and inconsistent assumptions. Sequentially finalizing sections can also produce incompatible conclusions because cross-concern relationships are not evaluated until too late.
The reporting-stage instruction is a common anti-pattern: it asks the final writer to cosmetically cover every topic without ensuring that the research workflow gathered enough evidence for each topic. Reliable multi-step workflows assign distinct items, preserve shared context, and require synthesis before final reporting.
Learn more about agent orchestration and tool-based workflows in the Agent SDK documentation and the Tool Use guide.

5.Scenario: Customer Support Resolution Agent You are building a customer support resolution agent using the Claude Agent SDK. The agent handles high-ambiguity requests like returns, billing disputes, and account issues. It has access to backend systems through MCP tools (get_customer, lookup_order, process_refund, escalate_to_human). Your target is 80%+ first-contact resolution while knowing when to escalate. Production logs show that lookup_order and process_refund both return the same failure text, "Operation failed." The agent retries declined refunds, tells customers to try again later when they provided invalid order IDs, and escalates temporary timeout cases that would likely succeed on retry. What change would best improve the agent's recovery decisions?

A. Route all refund and order lookup failures to escalate_to_human, avoiding autonomous recovery entirely after backend errors.

B. Return MCP tool errors with isError plus category, retryability, and safe messages distinguishing transient, validation, business, and permission failures.

C. Standardize every tool failure as a generic message, then ask Claude to infer recovery steps from conversation context.

D. Retry every failed backend tool call three times, then escalate unresolved cases without exposing error details to Claude.

Answer: B

Explanation:

Structured error responses let an agent make different decisions for different failure modes instead of treating every backend problem as the same event. In practice, an MCP tool should use an error signal such as isError and include metadata like errorCategory, isRetryable, and a human-readable explanation that is safe to show or summarize to the customer.

The underlying principle is that recovery logic depends on the nature of the failure: *transient* errors may be retried, *validation* errors usually require corrected input, *business* errors such as policy declines should be explained without retrying, and *permission* errors may require escalation or access handling. Generic failure text prevents Claude from selecting the right path and often causes wasted retries, premature escalation, or misleading customer responses.

Fixed retry counts are an anti-pattern when used as the primary response to all failures, because they ignore whether the error is actually retryable. Escalating every tool failure is also too conservative for an 80%+ first-contact resolution target, since many cases can be recovered locally with the right error details.

Learn more about MCP tool behavior in MCP Tools and Claude tool orchestration in Tool Use.