Vendor: Anthropic

Exam Code: CCAR-F

Exam Name: Claude Certified Architect - Foundations

Version: DEMO

QUESTION 1
You're implementing the escalation logic for when the agent should call escalate_to_human. Your team proposes four different approaches for triggering escalation. Which approach will most reliably identify cases that genuinely require human intervention?

A. Implement sentiment analysis that monitors for frustration indicators (negative language, repeated questions, exclamation marks) and trigger escalation when the frustration score exceeds a configured threshold.
B. Configure the agent to escalate after three consecutive tool calls that fail to resolve the customer's stated issue, ensuring a reasonable attempt before involving a human.
C. Instruct the agent to escalate when the customer requests a human, when the issue requires policy exceptions, or when the agent cannot make meaningful progress.
D. Build a rules engine that maps specific issue types, customer segments, and product categories to escalation decisions, removing the need for model judgment calls.

Answer: C
Explanation:
Escalation should be based on clear operational triggers: explicit user request, authority or policy limits, and lack of meaningful progress. This combines user intent, business constraints, and task outcome signals, making escalation more reliable than relying only on sentiment, fixed retry counts, or rigid rules.

QUESTION 2
Your order management system requires tools for three distinct operations: issuing refunds (requires amount and reason), canceling orders (requires reason), and requesting reshipments (requires shipping address). Each operation shares an order_id parameter but has different additional requirements. You notice during testing that with your current unified tool design, the agent frequently omits required parameters or includes irrelevant ones. What design change will most effectively improve parameter accuracy?

A. Keep one unified tool with a nested operation_details object parameter whose internal structure varies by operation type, documented in the tool description.
B. Keep one unified tool but add JSON Schema if-then-else conditionals to enforce that parameters like amount are required only when the operation type is "refund".
C. Split into three separate tools (issue_refund, cancel_order, request_reshipment), each defining only the parameters required for that specific operation.
D. Keep one unified tool with all parameters marked optional, but add detailed few-shot examples in the system prompt showing correct parameter combinations for each operation type.

Answer: C
Explanation:
Separate tools with operation-specific schemas make the required parameters clear and reduce ambiguity during tool selection. This improves parameter accuracy because each tool exposes only the fields needed for that exact action, such as refund amount and reason, cancellation reason, or reshipment address.

QUESTION 3
Your post_content tool requires user confirmation before publishing. The current workflow displays "Ready to post to social media. Confirm?" and analytics show users approve 98% of requests within 2 seconds. Post-mortems reveal incidents where posts went to wrong accounts, were scheduled for wrong times, or contained errors – all confirmed by users without catching the mistakes. How should you redesign the confirmation workflow?

A. Auto-approve routine posts and only require explicit confirmation for unusual patterns like posting to new accounts or large audiences
B. Require users to type a confirmation phrase instead of clicking a button
C. Add a mandatory waiting period before the confirm option becomes available
D. Include the complete post text, target account, scheduled time, and platform in the confirmation request

Answer: D
Explanation:
A confirmation step must present the specific action details the user is approving. Showing the full post content, target account, scheduled time, and platform helps users detect errors before execution, making confirmation meaningful rather than a generic approval click.

QUESTION 4
Your agent uses three tools: get_property_details(property_id) returns data including street address, get_price_history(property_id) returns historical pricing, and get_neighborhood_info(address) returns area statistics. You observe that get_neighborhood_info always requires get_property_details first just to extract the address, even when users specify the property by ID. This creates unnecessary latency and failure coupling – if the first call fails, the neighborhood request also fails. What tool design change best addresses this?

A. Create a lookup_address(property_id) helper tool for retrieving addresses.
B. Add retry logic and timeout handling to get_property_details.
C. Change get_neighborhood_info to accept property_id, resolving the address internally.
D. Consolidate into a single get_property_with_neighborhood(property_id) tool returning both datasets.

Answer: C
Explanation:
Allowing the neighborhood tool to accept the same identifier the user provides removes the need for the agent to chain through another tool just to extract an intermediate value. Resolving the address internally reduces latency, avoids unnecessary dependency on a prior tool call, and makes the workflow more reliable.

QUESTION 5
Your update_user_profile tool accepts a user_id (required) and an optional fields_to_update object. In testing, Claude frequently omits user_id or passes incorrectly structured data. What is most critical for helping Claude understand what parameter values to provide?

A. Strict JSON Schema type constraints marking user_id as required and defining fields_to_update as an object type
B. Verbose parameter names encoding format hints, such as user_id_string_uuid_format
C. Detailed error responses explaining why invalid parameter values were rejected
D. Clear parameter descriptions explaining expected format, such as “user_id: UUID of the user to update (required)”

Answer: A
Explanation:
Enforcing required fields and type constraints through a JSON Schema gives the agent clear rules about what parameters must be provided and their expected structure, preventing omissions or incorrectly formatted data.

QUESTION 6
Compliance requires that refunds exceeding $500 must automatically escalate to a human agent – this rule cannot be left to model discretion. Despite clear system prompt instructions, production logs show the agent occasionally processes high-value refunds directly (3% failure rate). How should you achieve guaranteed compliance?

A. Modify the refund tool to return an error with message “Amount exceeds policy limit – please escalate” when threshold is exceeded.
B. Add few-shot examples to the prompt showing correct escalation behavior at various refund amounts ($400, $500, $600).
C. Implement a hook to intercept tool calls; when the refund process amount exceeds $500, block it and invoke human escalation.
D. Strengthen the system prompt with emphatic language: “CRITICAL POLICY: Refunds over $500 MUST trigger human escalation. NEVER process these directly.”

Answer: C
Explanation:
Enforcing the policy outside of the model via a PreToolUse hook guarantees compliance. By intercepting high-value refund attempts and triggering human escalation, the system ensures the rule is applied consistently regardless of prompt instructions or agent behavior.

QUESTION 7
Production logs reveal inconsistent error handling: when lookup_order fails, the agent sometimes retries 5+ times (wasteful when the order ID doesn’t exist), sometimes escalates immediately (premature for temporary network issues), and sometimes asks users for clarification (inappropriate when the issue is a backend permission error). Investigation shows your MCP tool returns uniform error responses: {"isError": true, "content": [{"type": "text", "text": "Operation failed"}]}. The agent cannot distinguish between error types. What’s the most effective improvement?

A. Create an analyze_error MCP tool the agent calls after any failure to determine the error category and recommended action.
B. Add few-shot examples to the system prompt demonstrating how to interpret error message patterns and select appropriate responses for each.
C. Enhance error responses with structured metadata: include errorCategory (transient/validation/permission), isRetryable boolean, and a description of what caused the failure.
D. Implement retry logic with exponential backoff in your MCP server for all errors, returning to the agent only after retries are exhausted.

Answer: C
Explanation:
Structured error metadata gives the agent the information needed to choose the right recovery path. By distinguishing transient, validation, and permission failures, and by explicitly indicating whether retry is appropriate, the agent can avoid wasteful retries, premature escalation, and unnecessary user clarification.

QUESTION 8
When implementing your lookup_order MCP tool, the backend sometimes returns errors (e.g., “Order not found” or temporary database failures). What is the correct pattern for communicating these errors back to the agent?

A. Throw an exception from the tool handler so the agent framework can catch and log it
B. Return the error message in the tool result content with the isError flag set to true
C. Log the error server-side and return an empty result to avoid confusing the model
D. Return a success response with a “status” field indicating the error type

Answer: B
Explanation:
Returning structured tool results with isError set to true allows the agent to receive explicit error signals while preserving the error message. This enables the agent to reason about the failure, choose appropriate recovery actions, and provide meaningful feedback to the user.

QUESTION 9
Your process_refund tool returns two types of errors: technical errors (“503 Service Unavailable”, “Connection timeout”) that are transient (5% of calls), and business errors (“Order exceeds 30 day return window”, “Item already refunded”) that are permanent (12% of calls). Monitoring shows the agent wastes 3-4 turns retrying business errors that can never succeed. Currently, both error types return only a plain text message to Claude. What’s the most effective way to reduce wasted retries while improving customer-facing response quality?

A. Implement automatic retry logic at the tool level for technical errors only, passing business errors to Claude without retries.
B. Add few-shot examples showing how to distinguish retriable from non-retriable errors by parsing error message text.
C. Return structured error responses with retriable: false for business errors and a customer-friendly explanation for Claude to use.
D. Add a check_refund_eligibility tool that must be called before process_refund to prevent business rule violations.

Answer: C
Explanation:
Structured error responses let the agent distinguish permanent business-rule failures from transient technical failures without parsing plain text. Marking business errors as non-retriable and including a customer-friendly explanation prevents wasted retry turns while helping the agent give a clear, useful response to the customer.

Thank You for Trying Our Product

Lead2pass Certification Exam Features:

★ More than **99,900** Satisfied Customers Worldwide.

★ Average **99.9%** Success Rate.

★ **Free Update** to match latest and real exam scenarios.

★ **Instant Download** Access! No Setup required.

★ Questions & Answers are downloadable in **PDF** format and **VCE** test engine format.

★ Multi-Platform capabilities - **Windows, Laptop, Mac, Android, iPhone, iPod, iPad**.

★ **100%** Guaranteed Success or **100%** Money Back Guarantee.

★ **Fast**, helpful support **24x7**.

View list of all certification exams: http://www.lead2pass.com/all-products.html

10% Discount Coupon Code: **ASTR14**