WORKSHOP 2 OF 5

Tool Design & MCP Integration

Build clean tool interfaces Claude can reliably select and use

WHAT YOU'LL LEARN IN THIS WORKSHOP

- Write tool descriptions that eliminate selection ambiguity
- Return structured error responses with isError + metadata
- Scope tool access per agent to prevent cross-specialization misuse
- Configure project vs user MCP servers with .mcp.json
- Choose the right tool_choice mode: auto, any, or forced

EXAM WEIGHT: 18%	60 Questions | 120 Minutes | Pass Score: 720/1000

BEGINNER Foundations	INTERMEDIATE Application	ADVANCED Production

BEGINNER

1. Defining Tools — The Right Way

Tool descriptions are the primary signal Claude uses to decide which tool to call. Minimal or overlapping descriptions cause misrouting.

```
get_customer_tool = {
"name": "get_customer",
"description": (
"Retrieves a customer profile by customer ID. "
"Input: customer_id (format: CUST-XXXXXX). "
"Returns: {id, name, email, status, tier}. "
"Use BEFORE any order lookup or refund. "
"Do NOT use for: searching by name/email."
),
"input_schema": {
"type": "object",
"properties": {
"customer_id": {"type": "string", "pattern": "^CUST-[A-Z0-9]{6}$"}
},
"required": ["customer_id"]
}
}
```

INTERMEDIATE

2. Structured Error Responses & tool_choice

```
def process_refund(customer_id: str, amount: float) -> dict:
try:
result = payment_api.refund(customer_id, amount)
return {'success': True, 'refund_id': result.id}
except PaymentAPITimeout:
return {'isError': True, 'errorCategory': 'transient',
'isRetryable': True, 'message': 'Timeout — retry in 5s'}
except RefundPolicyViolation as e:
return {'isError': True, 'errorCategory': 'business_rule',
'isRetryable': False, 'message': str(e),
'recommendedAction': 'escalate_to_human'}
```

Mode	Behavior	When to Use
`"auto"`	Claude decides: tool or text	General purpose
`"any"`	Claude MUST call a tool	Force structured output
`{"type":"tool","name":"X"}`	Claude MUST call tool X	Force prerequisite step

ADVANCED

3. MCP Configuration & Tool Scoping

```
// .mcp.json — committed to version control
{
"mcpServers": {
"github": {
"command": "npx",
"args": ["-y", "@modelcontextprotocol/server-github"],
"env": { "GITHUB_TOKEN": "${GITHUB_TOKEN}" }
},
"jira": {
"command": "npx",
"args": ["-y", "@modelcontextprotocol/server-jira"],
"env": { "JIRA_URL": "${JIRA_URL}", "JIRA_TOKEN": "${JIRA_API_TOKEN}" }
}
}
}
```

```
# BAD: synthesis agent has all 12 tools
synthesis_agent = create_agent(tools=ALL_TOOLS)
# GOOD: each agent gets only role-relevant tools
search_agent = create_agent(tools=[web_search_tool, fetch_url_tool])
analysis_agent = create_agent(tools=[read_doc_tool, calculate_tool])
synthesis_agent = create_agent(tools=[verify_fact_tool])
```

Domain 2 — All 60 Practice Questions

TOTAL: 60 QUESTIONS	Beginner: 21	Intermediate: 27	Advanced: 12

BEGINNER

Q1. Why are tool descriptions critical for reliable tool selection?

A) Tool descriptions are documentation for developers — Claude doesn't read them

B) Tool descriptions are the primary signal Claude uses to decide which tool to call; vague descriptions cause misrouting

C) Longer descriptions always produce better tool selection

D) Claude selects tools based on their position in the tools array, not descriptions

Correct Answer: B

Claude reads tool descriptions to understand what each tool does and when to use it. Minimal or overlapping descriptions cause Claude to make incorrect selections. Every description must clearly state purpose, inputs, outputs, and scope boundaries.

Q2. What information should a well-written tool description include?

A) Only the tool name and return type

B) Purpose, expected input formats, example queries, edge cases, output format, when to use it, and when NOT to use it vs. similar tools

C) The implementation details of the tool function

D) Only a one-sentence summary — brevity is best for tool descriptions

Correct Answer: B

A complete tool description prevents all common misrouting scenarios: clear purpose prevents confusion with similar tools, input format guidance prevents invalid calls, negative examples ('do NOT use for X') set explicit boundaries.

Q3. What does the isError flag pattern in MCP tool responses accomplish?

A) It marks tool responses as errors so Claude knows to stop the agentic loop

B) It communicates structured failure information back to the agent so it can make intelligent recovery decisions

C) It triggers automatic retry logic in the Agent SDK

D) It logs the error to Anthropic's monitoring system

Correct Answer: B

isError: true in a tool result signals to Claude that the tool failed, not that the task is impossible. Combined with errorCategory, isRetryable, and a human-readable message, it enables Claude to choose the right recovery: retry, escalate, or try an alternative approach.

Q4. What are the four main error categories to distinguish in MCP tool error responses?

A) low, medium, high, critical

B) transient, validation, business_rule, permission

C) network, timeout, api, database

D) warning, error, fatal, unknown

Correct Answer: B

Transient (retry-able: timeouts, outages), validation (bad input: fix the request), business_rule (policy violation: different action needed), permission (access denied: no retry needed). Each category implies a different recovery strategy for Claude.

Q5. What does tool_choice='auto' mean?

A) Claude must call a tool — any tool in the list

B) Claude decides whether to call a tool or respond with plain text

C) Claude selects a tool automatically based on input format

D) Tool selection is determined by the Agent SDK, not Claude

Correct Answer: B

'auto' is the default — Claude reasons about whether calling a tool or responding with text is more appropriate. Use 'any' when you need to guarantee a tool is called, or force a specific tool with {'type':'tool','name':'...'} for prerequisites.

Q6. What is the difference between project-level and user-level MCP server configuration?

A) Project-level supports more tools; user-level has a tool limit of 10

B) Project-level (.mcp.json) is committed to version control and shared with the team; user-level (~/.claude.json) is personal and not shared

C) Project-level requires admin approval; user-level is self-service

D) They are identical in behavior — the difference is only naming convention

Correct Answer: B

Scope determines sharing: .mcp.json in the repo root is checked into git → every developer who clones the repo gets the same MCP tools. ~/.claude.json is personal → only you see those servers. Use project-level for team tools, user-level for experiments.

Q7. Why should you use environment variable expansion in .mcp.json instead of hardcoding tokens?

A) Environment variables load faster than hardcoded strings

B) Hardcoding API tokens in .mcp.json commits secrets to version control — env var expansion keeps secrets in the environment, not the repo

C) .mcp.json does not support string values — only env var references work

D) Environment variables enable automatic token rotation

Correct Answer: B

${TOKEN_NAME} in .mcp.json expands to the actual value at runtime from the developer's environment. The file itself contains no secret — it's safe to commit. Hardcoded tokens in version control create serious security vulnerabilities.

Q8. What is the Grep built-in tool best used for?

A) Finding files by name or extension pattern (e.g., **/*.test.ts)

B) Searching file contents for patterns like function names, error messages, or import statements

C) Reading entire file contents

D) Making targeted edits to specific lines in a file

Correct Answer: B

Grep is for content search — 'find all files that import from this module', 'find where this function is called'. Glob is for file name pattern matching (finding files by name/extension). Read/Write/Edit are for file content operations.

Q9. What is the Glob built-in tool best used for?

A) Searching file contents for a specific string or regex pattern

B) Finding files by name or path pattern (e.g., **/*.test.tsx, src/**/index.ts)

C) Reading all files in a directory recursively

D) Watching files for changes

Correct Answer: B

Glob matches file paths by pattern — use it to find all test files, all TypeScript files, or all index files in a specific directory structure. It returns paths, not content. Grep searches content within files.

Q10. When should you use the Edit tool instead of Read + Write?

A) Always use Edit — it is faster than Read + Write

B) Use Edit for targeted modifications when the text to replace is unique in the file; fall back to Read + Write when Edit cannot find a unique anchor

C) Use Edit for new file creation; Read + Write for modifications

D) Edit is for binary files; Read + Write is for text files

Correct Answer: B

Edit requires unique text matching — it finds a specific string and replaces it. If the target text appears multiple times, Edit may fail. Read the full file, modify in memory, Write back when Edit cannot identify a unique anchor point.

Q11. What does 'scoped tool access' mean in multi-agent design?

A) Limiting which users can invoke specific tools

B) Giving each agent only the tools relevant to its specific role, preventing cross-specialization misuse

C) Rate limiting tool calls to prevent API overuse

D) Encrypting tool inputs and outputs for security

Correct Answer: B

Scoped access: a synthesis agent gets only summarize and verify tools — not search or file-write tools. This prevents misuse (synthesis agent searching instead of synthesizing) and improves tool selection reliability within each agent.

Q12. What happens when you expose an MCP resource instead of requiring tool calls for data access?

A) Resources are more expensive than tool calls

B) Resources give the agent visibility into available data catalogs without requiring exploratory tool calls — reducing unnecessary round-trips

C) Resources can only contain text, not structured data

D) Resources are only available in project-scoped configurations

Correct Answer: B

MCP resources expose content catalogs (issue summaries, doc hierarchies, schemas) that the agent can browse without calling tools. Instead of calling list_issues() to discover what's available, the agent sees a resource with all issue summaries already available.

Q13. What is the recommended approach for standard integrations like Jira or GitHub in MCP?

A) Always build custom MCP servers for full control

B) Use existing community MCP servers; reserve custom server development for team-specific, non-standard workflows

C) Avoid MCP entirely for third-party integrations — use REST APIs directly

D) Build a universal MCP server that handles all third-party integrations

Correct Answer: B

The MCP ecosystem has mature community servers for common tools (GitHub, Jira, Slack, databases). Using them avoids reinventing the wheel. Build custom servers only for proprietary internal systems or workflows with unique requirements.

Q14. Why does a generic 'Operation failed' error response prevent intelligent recovery?

A) Claude cannot parse generic strings — only JSON is readable

B) Without error category, retryability flag, and context, Claude cannot distinguish between 'retry in 5 seconds' and 'this will never work — escalate instead'

C) Generic errors trigger the Agent SDK's automatic error handler

D) The isError flag is not set in generic errors, so Claude doesn't see them as failures

Correct Answer: B

Recovery intelligence requires: what type of error (transient/permanent?), is retry appropriate (isRetryable?), and what context helps recovery. 'Operation failed' provides none of this — Claude must guess, often poorly.

Q15. What is tool_choice='any' used for?

A) Letting Claude choose the best tool from multiple options

B) Guaranteeing that Claude calls a tool (any tool in the list) rather than responding with plain text

C) Selecting all tools in parallel

D) Disabling tool use for the current turn

Correct Answer: B

'any' is used when you need structured output and have multiple possible extraction schemas — you don't know which document type will be submitted, but you NEED a tool call (not a text response). Claude will call whichever tool it deems appropriate.

Q16. What does providing input/output examples in tool descriptions accomplish?

A) Examples increase the token count unnecessarily

B) Examples demonstrate expected behavior for edge cases and help Claude understand format expectations, reducing misuse

C) Examples override the input_schema — only one is needed

D) Examples are only useful for debugging, not production

Correct Answer: B

Examples in descriptions bridge the gap between abstract specification and practical behavior. 'Input: CUST-123456 → Output: {id, name, email, status}' is clearer than prose alone. Edge case examples (null handling, format variations) are especially valuable.

Q17. What does the tool_use_id field in a tool_result block do?

A) It identifies which API call generated the tool request

B) It links the tool result back to the specific tool_use block in Claude's response, enabling correct routing when multiple tools are called

C) It is a unique identifier for audit logging purposes

D) It specifies which Claude model processed the tool call

Correct Answer: B

When Claude requests multiple tools in one turn, each tool_use block has a unique id. Your tool_result must include the matching tool_use_id so Claude knows which result corresponds to which tool call. Mismatched IDs cause undefined behavior.

Q18. What makes a tool description 'boundary-explicit'?

A) The description is exactly 100 words

B) The description clearly states both what the tool DOES handle and what it does NOT handle, preventing misuse on out-of-scope inputs

C) The description includes a JSON schema with strict validation rules

D) The description is formatted with bullet points and headers

Correct Answer: B

'Do NOT use for: local files or database records' is a boundary statement. These negative boundaries are as important as positive capabilities — they prevent Claude from calling a web-scraping tool on a file path or a database tool on a URL.

Q19. When should you enhance an MCP tool's description beyond the default?

A) Never — default descriptions from community servers are always sufficient

B) When the default description causes Claude to prefer built-in tools (like Grep) over the more capable MCP tool for the use case

C) Only when the tool is newly created

D) When the tool is used by more than 3 agents

Correct Answer: B

Community MCP servers often have generic descriptions. If Claude consistently reaches for the Grep built-in instead of your powerful code-search MCP tool, enhance the MCP tool's description to explicitly explain its advantages over Grep for this context.

Q20. What problem does splitting a generic 'analyze_document' tool into 'extract_data_points', 'summarize_content', and 'verify_claim_against_source' solve?

A) It reduces API latency by distributing work across smaller tools

B) It eliminates tool selection ambiguity by giving each tool a specific, unambiguous purpose with clear input/output contracts

C) It allows three agents to call different tools simultaneously

D) It reduces the total number of tokens in the tool definitions

Correct Answer: B

A generic analyze_document tool forces Claude to guess which type of analysis is needed. Three purpose-specific tools with clear names and descriptions make the right choice obvious from context: need to extract structured data → extract_data_points; need a summary → summarize_content.

Q21. Why should tool names use verbs that describe the action rather than nouns that describe the data?

A) Verb-based names are shorter and save tokens in the tool definition

B) Verb names communicate what the tool DOES (get_customer, update_order, delete_record) enabling immediate selection clarity; noun names (customer, order, record) don't indicate the action type

C) Anthropic's API requires verb-prefixed tool names

D) Verb names enable automatic tool discovery without descriptions

Correct Answer: B

Action clarity: get_customer vs update_customer vs delete_customer are immediately distinguishable — the verb makes the operation unambiguous. 'customer' as a tool name tells you nothing about the operation. Verb-first naming is a best practice that reduces selection ambiguity.

INTERMEDIATE

Q22. You have two tools with similar functionality: search_knowledge_base (searches internal docs) and search_web (searches live internet). The agent consistently calls the wrong one. What is the best fix?

A) Remove one of the tools to eliminate ambiguity

B) Rename and rewrite both descriptions to explicitly state source type: 'Searches INTERNAL knowledge base (static, no internet)' vs 'Searches LIVE web (real-time, internet)'

C) Add a disambiguation instruction to the system prompt

D) Use tool_choice='any' to let Claude choose freely

Correct Answer: B

The root cause is overlapping descriptions. Make source type the primary differentiator in the name AND description. 'search_internal_kb' vs 'search_live_web' is immediately clear. Including 'internal/static' vs 'live/real-time' in descriptions eliminates ambiguity.

Q23. You need to enforce that extract_metadata is always called before any enrichment tools. Which approach is most reliable?

A) Add 'always call extract_metadata first' to the system prompt

B) Use tool_choice: {'type': 'tool', 'name': 'extract_metadata'} on the first API call, then switch to 'auto' for subsequent turns

C) Remove enrichment tools from the tool list and add them back after extraction

D) Create a combined extract_and_enrich tool that does both in sequence

Correct Answer: B

Forced tool selection on the first call guarantees extract_metadata runs first. Once metadata is extracted and in the conversation, subsequent turns with 'auto' allow Claude to choose appropriate enrichment tools. Prompt instructions for sequencing are unreliable.

Q24. A tool returns an empty array [] because no records matched the query. How should the tool response be structured?

A) {isError: true, message: 'No records found'} — empty results are an error

B) {data: [], total: 0, query: 'original query'} with isError absent/false — empty results are a successful query outcome

C) Return null to indicate no data

D) Throw an exception so Claude knows to retry with different parameters

Correct Answer: B

Empty results are not errors — they are valid successful query outcomes. Marking them as errors misleads Claude into thinking the tool failed and may need retrying. Return a success response with empty data, and include context about what was queried.

Q25. Your synthesis agent needs occasional fact-verification capability (cross-role tool). How should this be implemented without giving the synthesis agent full search access?

A) Add all search tools to the synthesis agent — it will only use what it needs

B) Create a constrained verify_fact tool that accepts a claim and returns a verification result, routing complex lookups through the coordinator

C) Route all fact verification through a dedicated verification subagent via coordinator

D) Use the coordinator to perform all fact verification on the synthesis agent's behalf

Correct Answer: B

Scoped cross-role tools solve this elegantly: a verify_fact tool (claim → verified/refuted/uncertain + source) gives synthesis capability without granting unrestricted search access. Complex lookups that exceed verify_fact's scope route back to the coordinator.

Q26. You are configuring a team's development environment where junior developers' MCP servers should have read-only access while senior developers have full access. How do you implement this with the MCP configuration system?

A) MCP doesn't support permission levels — everyone gets the same tools

B) Use separate .mcp.json configurations per role stored in different branches

C) Configure role-specific tool lists in the server; each developer uses the same server URL but different auth tokens that map to different permission levels

D) Create separate MCP server instances per role in the shared .mcp.json

Correct Answer: C

Permission levels are implemented server-side via auth tokens. The .mcp.json env var expansion (${DEVELOPER_TOKEN}) means each developer's environment has a different token that the MCP server maps to appropriate permissions. The client config is identical — the server handles authorization.

Q27. When should a subagent handle an error locally vs. propagate it to the coordinator?

A) Always propagate — the coordinator should handle all errors for centralized control

B) Handle locally: transient errors (retry with backoff). Propagate: errors that cannot be resolved locally (permission denied, permanent failures), always including what was attempted and any partial results

C) Always handle locally — never bother the coordinator with error details

D) Propagate all errors and let the coordinator decide whether to retry

Correct Answer: B

Local handling is appropriate for self-recoverable issues (temporary outages). Propagation is needed when the coordinator must make decisions (pivot strategy, escalate, use alternative source). Always include partial results when propagating — partial data is better than no data.

Q28. A developer asks why their MCP server configuration in .mcp.json requires GITHUB_TOKEN to be set on every developer's machine. What is the correct explanation?

A) This is a bug — .mcp.json should store the actual token

B) ${GITHUB_TOKEN} expands from the developer's local environment at runtime; this keeps the actual secret out of version control while sharing the configuration structure with the team

C) The token is only needed on CI machines — local developers can skip it

D) Tokens should be stored in CLAUDE.md as comments

Correct Answer: B

This is the intended security model: the configuration structure (which MCP server, which command, which env var name) is shared via git. The actual secret value lives only in each developer's local environment. No secret ever enters the repo.

Q29. What is the impact of adding a keyword like 'analyze' to a system prompt instruction when you also have a tool named 'analyze_document'?

A) No impact — system prompt keywords don't influence tool selection

B) Keyword-sensitive instructions in the system prompt can create unintended tool associations — the word 'analyze' may bias Claude toward calling analyze_document even when inappropriate

C) This improves tool selection by providing additional context

D) Tool names are case-insensitive so keyword matching doesn't apply

Correct Answer: B

System prompt language influences Claude's tool selection. If your prompt says 'always analyze the document structure first' and you have a tool called analyze_document, Claude may invoke it when the instruction was about a different type of analysis. Review system prompts for unintended keyword-tool associations.

Q30. You are designing tool interfaces for a financial data pipeline with 8 tools. Which grouping best follows the principle of scoped tool access?

A) Give all 8 tools to a single agent — simplicity over specialization

B) Data ingestion agent: fetch_raw_data, validate_schema. Analysis agent: calculate_metrics, detect_anomalies. Reporting agent: format_report, send_email, archive_results

C) Split tools evenly: 4 tools per agent regardless of functional relationship

D) Give each tool to its own dedicated agent

Correct Answer: B

Functional cohesion: each agent gets tools that work together for its specific role. Ingestion doesn't need reporting tools. Reporting doesn't need data fetching tools. This grouping minimizes cross-role misuse and makes each agent's purpose clear.

Q31. How should a tool return a partial success when it completes 7 of 10 requested operations before hitting a rate limit?

A) Return isError: true — partial completion is a failure

B) Return {partial_success: true, completed: [7 items], failed: [3 items with error details], retry_after: 30, message: 'Rate limited after 7 operations'}

C) Return only the 7 successful results — the agent will retry for the rest

D) Return isError: true only if 0 operations completed; success if any completed

Correct Answer: B

Partial success is real information. The agent needs to know: what succeeded (don't re-do), what failed (retry only these), and when to retry (retry_after). Returning only the 7 results causes the agent to think the task is complete and miss the 3.

Q32. When MCP tools are added to a Claude Code session, when are they discovered and made available?

A) On-demand, when Claude first decides to call a tool in that category

B) At connection time when the MCP servers start — all tools from all configured servers are discovered simultaneously and available throughout the session

C) They must be explicitly enabled per session with a /tools command

D) Tools are discovered lazily on first call and cached for subsequent calls

Correct Answer: B

MCP tool discovery happens at session start. All configured servers connect, all tools are enumerated, and they become part of the agent's available tool set. There is no per-call discovery overhead.

Q33. Your agent has a tool that fetches customer data and occasionally returns PII. You want to ensure PII never appears in Claude's context. What is the correct approach?

A) Add 'never store PII in memory' to the system prompt

B) Implement a PostToolUse hook that detects and redacts PII patterns before the result reaches Claude

C) Use a separate privacy-aware Claude model

D) Instruct the tool to omit PII fields before returning

Correct Answer: B

PostToolUse hooks provide deterministic PII redaction — they run on every tool result before Claude sees it. System prompt instructions are probabilistic. The tool itself may be an external system you can't modify. Hook-based redaction is the only reliable architectural solution.

Q34. What is the recommended strategy for building codebase understanding incrementally using built-in tools?

A) Read all files upfront to build complete context before starting analysis

B) Start with Grep to find entry points and key definitions, then use Read to follow imports and trace specific execution paths — discovering depth on demand

C) Use Glob to find all files, then Read each one in alphabetical order

D) Start with a Bash command to print the entire codebase

Correct Answer: B

Incremental discovery: Grep reveals structure (where is the main function? what imports X?), Read follows specific paths of interest. This is more efficient than reading all files upfront and avoids context window exhaustion on large codebases.

Q35. You notice that Claude is using a built-in Grep tool instead of your MCP-based semantic code search tool, even though the MCP tool produces much better results. What is the likely cause and fix?

A) Built-in tools are always preferred over MCP tools by design

B) The MCP tool's description doesn't clearly explain its advantage over Grep for code search; enhance the description to explain semantic search capabilities and when it outperforms Grep

C) Increase the weight of the MCP tool in the tool configuration

D) Remove the Grep built-in from allowedTools

Correct Answer: B

Tool selection is description-driven. If Grep's description says 'search file contents for patterns' and your MCP tool says 'search tool for code', Claude reasonably prefers the more specific description. Make your MCP description explicitly better: 'Semantic code search — understands intent, not just literal matches; use instead of Grep for conceptual searches'.

Q36. A tool that deletes records must require confirmation before executing in production. How should this be implemented architecturally?

A) Add 'always confirm before deleting' to the tool description

B) Implement a PreToolUse hook that intercepts delete_record tool calls and requires a human_approval field to be true before allowing execution

C) Create a confirm_delete tool that Claude must call before delete_record

D) Use a two-step system prompt that asks Claude to verify before deleting

Correct Answer: B

Programmatic enforcement via hook: block delete_record unless human_approval=true in the tool inputs. This cannot be bypassed by prompt injection or model non-compliance. Option C (confirm_delete tool) is also good but Claude could theoretically skip it; a hook cannot be skipped.

Q37. How should you structure MCP tool error responses for a permission denied error?

A) {isError: true, message: 'Access denied'} — brief is best

B) {isError: true, errorCategory: 'permission', isRetryable: false, message: 'Agent lacks write access to this resource', requiredPermission: 'billing:write', escalationPath: 'request access via IT portal'}

C) Throw an exception with the error code

D) Return empty data with a comment field noting the permission issue

Correct Answer: B

Rich permission errors enable intelligent handling: isRetryable: false (don't waste time retrying), requiredPermission tells Claude exactly what's needed, escalationPath provides an actionable next step. Claude can then explain to the user what access needs to be requested.

Q38. What is the purpose of the allowedTools restriction in subagent configuration?

A) To limit API costs by reducing the number of available tools

B) To ensure each subagent has access only to tools relevant to its specific role, preventing accidental or intentional misuse of out-of-scope tools

C) To comply with Anthropic's API terms of service

D) To prevent tool name collisions between different MCP servers

Correct Answer: B

allowedTools enforces the principle of least privilege for agents. A web search subagent with database write tools is a risk — it might accidentally (or via prompt injection) modify data. Restricting tools to role-relevant ones minimizes blast radius and improves reliability.

Q39. Your team has both a Jira MCP server (community) and a custom internal ticket system MCP server. How should these be configured?

A) Use only one ticket system — two creates confusion

B) Configure both in project-level .mcp.json; write clear tool descriptions that distinguish when to use each (e.g., 'use Jira for customer-facing tickets, use internal system for engineering tasks')

C) Put the community Jira server in user-level config and the custom server in project-level

D) Combine them into a single MCP proxy server

Correct Answer: B

Both can coexist in .mcp.json. The key is disambiguation via descriptions: Claude needs to understand when to use each. Clear 'use for' criteria in tool descriptions prevent Claude from guessing which ticket system to use.

Q40. What is the correct approach when the Edit tool fails to find a unique match in a file?

A) Use a more specific pattern in the Edit call

B) Read the full file content, apply the modification programmatically in your code, then Write the modified content back

C) Skip the modification — if Edit fails, the file cannot be modified

D) Use Bash with sed to make the edit instead

Correct Answer: B

Read + modify + Write is the reliable fallback when Edit cannot find a unique anchor. Read gives you the full file content, you apply the change, Write replaces the file. This is always reliable regardless of uniqueness constraints.

Q41. An MCP tool result contains a list of 500 items that exceeds practical context limits. What is the best tool design response?

A) Return all 500 items — context limits are the caller's problem

B) Return a paginated response with a cursor: {data: [first 20], total: 500, cursor: 'abc123', has_more: true} and a corresponding fetch_next tool

C) Return only the top 10 items silently

D) Return an error indicating the result is too large

Correct Answer: B

Pagination at the tool level is the right design: return manageable chunks, signal there's more (has_more + cursor), and provide a way to get the next page. Silent truncation is deceptive; erroring on large results is unhelpful. Pagination enables incremental consumption.

Q42. A tool receives an input that passes schema validation but will cause the underlying system to fail (e.g., a customer_id that doesn't exist in the database). How should this be handled?

A) The schema should be extended to validate all business rules

B) Return a structured not_found error: {isError: true, errorCategory: 'not_found', message: 'Customer CUST-999 does not exist', isRetryable: false, suggestion: 'Verify the customer ID or use search_customer to find the correct ID'}

C) Return an empty result as if the customer exists but has no data

D) Throw a validation exception before attempting the database lookup

Correct Answer: B

Business logic errors beyond schema validation need structured error responses. not_found is a distinct category from validation or transient errors — it's not retryable as-is, but it suggests a concrete alternative action (search to find the correct ID). This enables intelligent Claude recovery.

Q43. What is the benefit of exposing MCP resources (like a list of available reports) versus requiring tool calls to discover available data?

A) Resources are faster than tool calls due to caching

B) Resources give the agent a browsable catalog of what's available without consuming tool call budget or adding latency — the agent can plan efficiently before fetching

C) Resources have higher API rate limits than tool calls

D) Resources are only readable, preventing accidental writes

Correct Answer: B

Discoverability without tool call cost: an MCP resource listing 'Available reports: Q1-2024-revenue, Q2-2024-revenue...' lets the agent understand what's available before deciding what to fetch. Without this, the agent must call list_reports() (a tool call) just to discover options.

Q44. You have a tool that returns a deeply nested JSON object (8 levels deep) with 200 fields. Claude frequently references the wrong fields. How do you fix this?

A) Flatten the JSON to a single level before returning

B) Add a summary field at the top level: {summary: {key_field_1, key_field_2, key_field_3}, full_data: {...nested...}} — Claude reads summary first for the most important data

C) Return only the most important 10 fields

D) Document the structure in the tool description

Correct Answer: B

Attention structure: Claude's attention is strongest on data it encounters first. A top-level summary with the most important fields ensures reliable access to critical data. The full nested structure is preserved for cases where Claude needs to drill deeper. Summary-first design beats documentation alone.

Q45. A tool call to an external payment API hangs for 30 seconds before timing out. How should the tool handle and communicate this?

A) Wait indefinitely — never timeout tool calls

B) Implement a client-side timeout (e.g., 10 seconds); return {isError: true, errorCategory: 'transient', isRetryable: true, message: 'Payment API timeout after 10s', retry_after: 30, attempted_amount: amount}

C) Retry 3 times before returning an error

D) Return a success response with pending status

Correct Answer: B

Client-side timeouts prevent hung tool calls from blocking the agentic loop indefinitely. Structured timeout errors with retry_after give Claude the information to make a retry decision. The attempted_amount field prevents Claude from losing track of what was being processed during recovery.

Q46. You need to design a tool that processes images for a Claude agent. What is the correct way to return image analysis results?

A) Return the raw image bytes as base64 in the tool result

B) Return structured analysis data: {objects_detected: [...], text_extracted: '...', dimensions: {w, h}, confidence_scores: {...}} — Claude receives structured data, not raw image bytes

C) Return a URL where Claude can fetch the analyzed image

D) Return a human-readable description paragraph

Correct Answer: B

Tools should return structured data that Claude can reason about programmatically. Raw image bytes in a tool result serve no purpose — Claude cannot process them differently than the original. Structured analysis (detected objects, extracted text, metadata) is actionable and can be schema-validated.

Q47. What is the relationship between tool input_schema and the quality of Claude's tool calls?

A) Schema is only for validation — it doesn't affect the quality of Claude's inputs

B) Detailed input_schema with property descriptions, examples, format patterns, and constraints directly guides Claude's input construction, reducing invalid or suboptimal tool calls

C) Simpler schemas produce better tool calls — complexity confuses the model

D) Schema properties are developer documentation only — Claude doesn't read them

Correct Answer: B

Claude reads and uses property descriptions in input_schema: a property with description: 'Customer ID in format CUST-XXXXXX, e.g., CUST-ABC123' produces correctly formatted IDs. A property with description: 'id' produces guessed formats. Schema quality directly affects tool call quality.

Q48. You discover that Claude sometimes calls your search tool with natural language queries when the tool expects keyword-based queries. How do you fix this?

A) Update the model to handle natural language

B) Add explicit input guidance to the tool description and property description: 'Accepts keyword-based queries only. NOT natural language. Good: "python async error handling". Bad: "How do I handle async errors in Python?"'

C) Pre-process Claude's tool inputs to convert natural language to keywords

D) Use a different tool naming convention

Correct Answer: B

Input format guidance with good/bad examples directly in the description eliminates the natural language vs keyword confusion. The contrast example ('Good: ... Bad: ...') is especially effective — it shows the difference concretely rather than describing it abstractly.

ADVANCED

Q49. You are designing tool interfaces for a healthcare data system. Which approach best balances Claude's tool selection reliability with security requirements?

A) Give Claude all 25 tools and rely on the system prompt for access control

B) Implement role-based tool sets with a maximum of 5-7 tools per agent role; enforce data access controls in PreToolUse hooks; never rely on prompt instructions for PHI access control

C) Use a single universal tool with a permission parameter

D) Separate tools for reading vs. writing are sufficient security

Correct Answer: B

Healthcare requires defense in depth: (1) scoped tool sets improve selection reliability and reduce attack surface, (2) PreToolUse hooks provide programmatic PHI access control that cannot be bypassed, (3) never rely on prompts for HIPAA compliance. This is the pattern the exam rewards.

Q50. You discover that a malformed input to one of your MCP tools causes it to expose internal system paths in error messages. How should you fix this?

A) Add input validation instructions to the tool description

B) Implement input sanitization in the tool itself and ensure error messages never contain internal paths, stack traces, or system details — return generic error categories instead

C) Use a PreToolUse hook to validate inputs before calling the tool

D) Ask Claude not to pass malformed inputs in the system prompt

Correct Answer: B

Security must be implemented at the tool layer: sanitize inputs before processing, and ensure errors never leak internal details. PreToolUse hooks can add validation, but the tool itself must also be hardened since it may be called from contexts outside your agent.

Q51. You need to implement a tool that calls an external API with rate limits of 100 requests per minute. Your agentic system can generate up to 500 tool calls per minute. What is the correct architectural approach?

A) Let tool calls fail with rate limit errors and rely on Claude to retry

B) Implement a token bucket rate limiter in the tool wrapper; return {isError: true, errorCategory: 'transient', retry_after: seconds} when rate limited; batch multiple tool inputs where the API supports it

C) Add 'do not call this tool more than 100 times per minute' to the tool description

D) Route tool calls through a queue with no back-pressure mechanism

Correct Answer: B

Token bucket rate limiting at the tool layer provides accurate rate control. Returning structured retry_after enables Claude to time retries correctly. Batching reduces individual API calls. Prompt-based rate limiting is unreliable — Claude doesn't track real-time call frequency.

Q52. A multi-tenant system has customers A and B whose data must never mix. Both use the same agent system with the same tools. How do you implement data isolation?

A) Separate the tool code for each customer

B) Pass tenant_id in all tool calls; implement tenant isolation in the MCP server layer with row-level security or separate databases; validate tenant_id in PreToolUse hooks against the authenticated session

C) Use separate Claude sessions and trust that sessions are isolated

D) Add 'only access data for the current customer' to the system prompt

Correct Answer: B

Multi-tenant isolation requires: (1) tenant context in every tool call, (2) server-side enforcement (RLS or separate schemas), (3) PreToolUse hook validation that Claude cannot access another tenant's data. Prompt-only isolation is a security vulnerability.

Q53. You need to design a tool ecosystem for an agent that processes legal documents. The tools include: extract_entities, verify_citations, check_jurisdiction, calculate_deadlines, file_document, send_notification, archive_document, update_case. How should these be distributed across agents?

A) One agent with all 8 tools

B) Analysis agent (extract_entities, verify_citations, check_jurisdiction, calculate_deadlines); Filing agent (file_document, send_notification, archive_document, update_case) — with coordinator managing handoff and requiring human approval before filing

C) One tool per agent (8 agents)

D) Read tools in one agent, write tools in another

Correct Answer: B

Functional cohesion + safety: analysis tools belong together (they work on document content), filing tools belong together (they act on external systems). Separating analysis from filing creates a natural approval gate — the coordinator reviews analysis results before delegating to the filing agent.

Q54. You are building a tool that executes code snippets. What security controls must be implemented at the tool layer regardless of Claude's instructions?

A) Add 'only execute safe code' to the tool description

B) Implement: sandboxed execution environment (no network access, no file system access outside designated directory), resource limits (CPU time, memory), timeout enforcement, and output sanitization before returning to Claude

C) Validate code syntax before execution

D) Use a separate Claude instance to review code before execution

Correct Answer: B

Code execution tools require hardened sandboxing at the infrastructure level. Prompt instructions cannot prevent malicious code execution. Sandboxing, resource limits, and timeouts must be enforced by the execution environment itself, independent of any LLM instructions.

Q55. Your MCP server needs to expose a large corporate knowledge base (10,000 documents) to Claude agents. How should you design the interface for maximum efficiency?

A) Return all 10,000 documents as a single tool response

B) Expose a semantic_search tool for targeted queries, an MCP resource listing document categories/summaries, and paginated fetch tools — enabling agents to discover and retrieve only relevant content

C) Create one tool per document

D) Compress all documents into a single 200K-token context injection

Correct Answer: B

Multi-tier access: MCP resources provide structure visibility (what's available), semantic_search enables targeted retrieval, paginated fetch handles large results. This avoids overwhelming Claude with irrelevant content while making the full knowledge base accessible on demand.

Q56. A PostToolUse hook must handle results from tools that can return up to 50MB of data. Some agents need full data; others need summaries. How should the hook be designed?

A) Always return full data — let each agent decide what to use

B) The hook receives the calling agent's context; for agents with 'summary_mode' configuration, apply compression/summarization before returning; for full-access agents, return complete data

C) Always return summaries — 50MB is too large for any context

D) Implement separate tool endpoints for summary vs. full access

Correct Answer: B

Context-aware hooks adapt transformation to the consumer. Knowing which agent is calling (from the hook's execution context) enables appropriate data transformation. Summary agents get compressed data; analysis agents get full data. One hook handles both cases.

Q57. You are designing the tool interface for an agent that needs to interact with a database. The database has 50 tables. What is the most reliable tool design?

A) One universal query_database tool that accepts raw SQL

B) Role-specific semantic tools: get_customer(id), get_orders(customer_id, date_range), update_order_status(order_id, status) — named by business intent, not technical implementation

C) One tool per database table

D) Two tools: read_database and write_database with SQL parameters

Correct Answer: B

Semantic tools beat raw SQL: (1) Claude doesn't need to know the schema, (2) business-intent names eliminate ambiguity, (3) specific parameter schemas prevent SQL injection, (4) you control exactly what operations are possible. Raw SQL is a security risk and selection reliability nightmare.

Q58. What is the most robust way to ensure tool call ordering when you have three tools that must execute in sequence: authenticate → fetch_data → transform_data?

A) List them in the correct order in the tools array

B) Combine multiple enforcement layers: forced tool_choice for authenticate, a PreToolUse hook that blocks fetch_data until authentication token exists in context, and another that blocks transform_data until raw data is present

C) Add sequencing instructions to each tool's description

D) Use a single combined authenticate_fetch_transform tool

Correct Answer: B

Multi-layer enforcement: forced first step ensures the sequence starts correctly, prerequisite hooks ensure each step only runs after dependencies are satisfied. This is defense in depth for workflow ordering — each layer adds reliability.

Q59. When multiple agents share the same MCP server but need different levels of access, what is the correct pattern?

A) Configure separate MCP server instances for each access level

B) Use authentication tokens that map to permission levels in the MCP server; pass the appropriate token per agent context; the server enforces access control at the tool call level

C) Define separate tool sets for each agent in the MCP server

D) Use a proxy layer that filters tool calls based on agent identity

Correct Answer: B

Token-based authorization at the MCP server level is the correct pattern: one server instance, multiple permission levels via auth tokens. The coordinator provides each subagent's context with the appropriate token. This is scalable (one server, many permission levels) and the server enforces access control, not the agent.

Q60. A tool that writes files accepts a path parameter. What validation should the tool perform to prevent path traversal attacks?

A) Add path: string to the input schema — schema validation is sufficient

B) Server-side validation: normalize the path, verify it resolves within the allowed directory, reject any path containing '../', absolute paths outside allowed directories, or symlinks pointing outside

C) Add 'only write to the project directory' to the tool description

D) Use a sandbox environment and allow any path

Correct Answer: B

Path traversal is a security vulnerability where '../../../etc/passwd' escapes the intended directory. Schema validation only checks type, not content security. Server-side path normalization and directory boundary checking is the correct security control. Prompt instructions cannot prevent this attack.