

/doc-coauthoring

Source: `~/ .claude/skills/doc-coauthoring/SKILL.md`

name: doc-coauthoring **description:**
Guide users through a structured workflow for co-authoring documentation. Use when user wants to write documentation, proposals, technical specs, decision docs, or similar structured content. This workflow helps users efficiently transfer context, refine content through iteration, and verify the doc works for readers. Trigger when user mentions writing docs, creating proposals, drafting specs, or similar documentation tasks.

Doc Co-Authoring Workflow

This skill provides a structured workflow for guiding users through collaborative document creation. Act as an active guide, walking users through three stages: Context Gathering, Refinement & Structure, and Reader Testing.

When to Offer This Workflow

Trigger conditions:

- User mentions writing documentation: "write a doc", "draft a proposal", "create a spec", "write up"
- User mentions specific doc types: "PRD", "design doc", "decision doc", "RFC"
- User seems to be starting a substantial writing task

Initial offer: Offer the user a structured workflow for co-authoring the document. Explain the three stages:

1. **Context Gathering:** User provides all relevant context while Claude asks clarifying questions
2. **Refinement & Structure:** Iteratively build each section through brainstorming and editing
3. **Reader Testing:** Test the doc with a fresh Claude (no context) to catch blind spots before others read it

Explain that this approach helps ensure the doc works well when others read it (including when they paste it into Claude). Ask if they want to try this workflow or prefer to work freeform.

If user declines, work freeform. If user accepts, proceed to Stage 1.

Stage 1: Context Gathering

Goal: Close the gap between what the user knows and what Claude knows, enabling smart guidance later.

Initial Questions

Start by asking the user for meta-context about the document:

1. What type of document is this? (e.g., technical spec, decision doc, proposal)
2. Who's the primary audience?

3. What's the desired impact when someone reads this?
4. Is there a template or specific format to follow?
5. Any other constraints or context to know?

Inform them they can answer in shorthand or dump information however works best for them.

If user provides a template or mentions a doc type:

- Ask if they have a template document to share
- If they provide a link to a shared document, use the appropriate integration to fetch it
- If they provide a file, read it

If user mentions editing an existing shared document:

- Use the appropriate integration to read the current state
- Check for images without alt-text
- If images exist without alt-text, explain that when others use Claude to understand the doc, Claude won't be able to see them. Ask if they want alt-text generated. If so, request they paste each image into chat for descriptive alt-text generation.

Info Dumping

Once initial questions are answered, encourage the user to dump all the context they have.

Request information such as:

- Background on the project/problem
- Related team discussions or shared documents
- Why alternative solutions aren't being used
- Organizational context (team dynamics, past incidents, politics)
- Timeline pressures or constraints
- Technical architecture or dependencies
- Stakeholder concerns

Advise them not to worry about organizing it - just get it all out. Offer multiple ways to provide context:

- Info dump stream-of-consciousness
- Point to team channels or threads to read
- Link to shared documents

If integrations are available (e.g., Slack, Teams, Google Drive, SharePoint, or other MCP servers), mention that these can be used to pull in context directly.

If no integrations are detected and in Claude.ai or Claude app: Suggest they can enable connectors in their Claude settings to allow pulling context from messaging apps and document storage directly.

Inform them clarifying questions will be asked once they've done their initial dump.

During context gathering:

- If user mentions team channels or shared documents:
 - If integrations available: Inform them the content will be read now, then use the appropriate integration
 - If integrations not available: Explain lack of access. Suggest they enable connectors in Claude settings, or paste the relevant content directly.
- If user mentions entities/projects that are unknown:
 - Ask if connected tools should be searched to learn more
 - Wait for user confirmation before searching
- As user provides context, track what's being learned and what's still unclear

Asking clarifying questions:

When user signals they've done their initial dump (or after substantial context provided), ask clarifying questions to ensure understanding:

Generate 5-10 numbered questions based on gaps in the context.

Inform them they can use shorthand to answer (e.g., "1: yes, 2: see #channel, 3: no because backwards compat"), link to more docs, point to channels to read, or just keep info-dumping. Whatever's most efficient for them.

Exit condition: Sufficient context has been gathered when questions show understanding - when edge cases and trade-offs can be asked about without needing basics explained.

Transition: Ask if there's any more context they want to provide at this stage, or if it's time to move on to drafting the document.

If user wants to add more, let them. When ready, proceed to Stage 2.

Stage 2: Refinement & Structure

Goal: Build the document section by section through brainstorming, curation, and iterative refinement.

Instructions to user: Explain that the document will be built section by section. For each section:

1. Clarifying questions will be asked about what to include
2. 5-20 options will be brainstormed
3. User will indicate what to keep/remove/combine
4. The section will be drafted
5. It will be refined through surgical edits

Start with whichever section has the most unknowns (usually the core decision/proposal), then work through the rest.

Section ordering:

If the document structure is clear: Ask which section they'd like to start with.

Suggest starting with whichever section has the most unknowns. For decision docs, that's usually the core proposal. For specs, it's typically the technical approach. Summary sections are best left for last.

If user doesn't know what sections they need: Based on the type of document and template, suggest 3-5 sections appropriate for the doc type.

Ask if this structure works, or if they want to adjust it.

Once structure is agreed:

Create the initial document structure with placeholder text for all sections.

If access to artifacts is available: Use `create_file` to create an artifact. This gives both Claude and the user a scaffold to work from.

Inform them that the initial structure with placeholders for all sections will be created.

Create artifact with all section headers and brief placeholder text like "[To be written]" or "[Content here]".

Provide the scaffold link and indicate it's time to fill in each section.

If no access to artifacts: Create a markdown file in the working directory. Name it appropriately (e.g., `decision-doc.md`, `technical-spec.md`).

Inform them that the initial structure with placeholders for all sections will be created.

Create file with all section headers and placeholder text.

Confirm the filename has been created and indicate it's time to fill in each section.

For each section:

Step 1: Clarifying Questions

Announce work will begin on the [SECTION NAME] section. Ask 5-10 clarifying questions about what should be included:

Generate 5-10 specific questions based on context and section purpose.

Inform them they can answer in shorthand or just indicate what's important to cover.

Step 2: Brainstorming

For the [SECTION NAME] section, brainstorm [5-20] things that might be included, depending on the section's complexity. Look for:

- Context shared that might have been forgotten
- Angles or considerations not yet mentioned

Generate 5-20 numbered options based on section complexity. At the end, offer to brainstorm more if they want additional options.

Step 3: Curation

Ask which points should be kept, removed, or combined. Request brief justifications to help learn priorities for the next sections.

Provide examples:

- "Keep 1,4,7,9"
- "Remove 3 (duplicates 1)"
- "Remove 6 (audience already knows this)"
- "Combine 11 and 12"

If user gives freeform feedback (e.g., "looks good" or "I like most of it but...") instead of numbered selections, extract their preferences and proceed. Parse what they want kept/removed/changed and apply it.

Step 4: Gap Check

Based on what they've selected, ask if there's anything important missing for the [SECTION NAME] section.

Step 5: Drafting

Use `str_replace` to replace the placeholder text for this section with the actual drafted content.

Announce the [SECTION NAME] section will be drafted now based on what they've selected.

If using artifacts: After drafting, provide a link to the artifact.

Ask them to read through it and indicate what to change. Note that being specific helps learning for the next sections.

If using a file (no artifacts): After drafting, confirm completion.

Inform them the [SECTION NAME] section has been drafted in [filename]. Ask them to read through it and indicate what to change. Note that being specific helps learning for the next sections.

Key instruction for user (include when drafting the first section): Provide a note: Instead of editing the doc directly, ask them to indicate what to change. This helps learning of their style for future sections. For example: "Remove the X bullet - already covered by Y" or "Make the third paragraph more concise".

Step 6: Iterative Refinement

As user provides feedback:

- Use `str_replace` to make edits (never reprint the whole doc)
- **If using artifacts:** Provide link to artifact after each edit
- **If using files:** Just confirm edits are complete
- If user edits doc directly and asks to read it: mentally note the changes they made and keep them in mind for future sections (this shows their preferences)

Continue iterating until user is satisfied with the section.

Quality Checking

After 3 consecutive iterations with no substantial changes, ask if anything can be removed without losing important information.

When section is done, confirm [SECTION NAME] is complete. Ask if ready to move to the next section.

Repeat for all sections.

Near Completion

As approaching completion (80%+ of sections done), announce intention to re-read the entire document and check for:

- Flow and consistency across sections
- Redundancy or contradictions
- Anything that feels like "slop" or generic filler
- Whether every sentence carries weight

Read entire document and provide feedback.

When all sections are drafted and refined: Announce all sections are drafted. Indicate intention to review the complete document one more time.

Review for overall coherence, flow, completeness.

Provide any final suggestions.

Ask if ready to move to Reader Testing, or if they want to refine anything else.

Stage 3: Reader Testing

Goal: Test the document with a fresh Claude (no context bleed) to verify it works for readers.

Instructions to user: Explain that testing will now occur to see if the document actually works for readers. This catches blind spots - things that make sense to the authors but might confuse others.

Testing Approach

If access to sub-agents is available (e.g., in Claude Code):

Perform the testing directly without user involvement.

Step 1: Predict Reader Questions

Announce intention to predict what questions readers might ask when trying to discover this document.

Generate 5-10 questions that readers would realistically ask.

Step 2: Test with Sub-Agent

Announce that these questions will be tested with a fresh Claude instance (no context from this conversation).

For each question, invoke a sub-agent with just the document content and the question.

Summarize what Reader Claude got right/wrong for each question.

Step 3: Run Additional Checks

Announce additional checks will be performed.

Invoke sub-agent to check for ambiguity, false assumptions, contradictions.

Summarize any issues found.

Step 4: Report and Fix

If issues found: Report that Reader Claude struggled with specific issues.

List the specific issues.

Indicate intention to fix these gaps.

Loop back to refinement for problematic sections.

If no access to sub-agents (e.g., claude.ai web interface):

The user will need to do the testing manually.

Step 1: Predict Reader Questions

Ask what questions people might ask when trying to discover this document. What would they type into Claude.ai?

Generate 5-10 questions that readers would realistically ask.

Step 2: Setup Testing

Provide testing instructions:

1. Open a fresh Claude conversation: <https://claude.ai>
2. Paste or share the document content (if using a shared doc platform with connectors enabled, provide the link)
3. Ask Reader Claude the generated questions

For each question, instruct Reader Claude to provide:

- The answer
- Whether anything was ambiguous or unclear
- What knowledge/context the doc assumes is already known

Check if Reader Claude gives correct answers or misinterprets anything.

Step 3: Additional Checks

Also ask Reader Claude:

- "What in this doc might be ambiguous or unclear to readers?"
- "What knowledge or context does this doc assume readers already have?"
- "Are there any internal contradictions or inconsistencies?"

Step 4: Iterate Based on Results

Ask what Reader Claude got wrong or struggled with. Indicate intention to fix those gaps.

Loop back to refinement for any problematic sections.

Exit Condition (Both Approaches)

When Reader Claude consistently answers questions correctly and doesn't surface new gaps or ambiguities, the doc is ready.

Final Review

When Reader Testing passes: Announce the doc has passed Reader Claude testing. Before completion:

1. Recommend they do a final read-through themselves - they own this document and are responsible for its quality
2. Suggest double-checking any facts, links, or technical details
3. Ask them to verify it achieves the impact they wanted

Ask if they want one more review, or if the work is done.

If user wants final review, provide it. Otherwise: Announce document completion. Provide a few final tips:

- Consider linking this conversation in an appendix so readers can see how the doc was developed
- Use appendices to provide depth without bloating the main doc
- Update the doc as feedback is received from real readers

Tips for Effective Guidance

Tone:

- Be direct and procedural
- Explain rationale briefly when it affects user behavior
- Don't try to "sell" the approach - just execute it

Handling Deviations:

- If user wants to skip a stage: Ask if they want to skip this and write freeform
- If user seems frustrated: Acknowledge this is taking longer than expected. Suggest ways to move faster
- Always give user agency to adjust the process

Context Management:

- Throughout, if context is missing on something mentioned, proactively ask
- Don't let gaps accumulate - address them as they come up

Artifact Management:

- Use `create_file` for drafting full sections
- Use `str_replace` for all edits
- Provide artifact link after every change
- Never use artifacts for brainstorming lists - that's just conversation

Quality over Speed:

- Don't rush through stages
- Each iteration should make meaningful improvements
- The goal is a document that actually works for readers

Revision #5

Created 2026-02-18 08:39:53 UTC by John

Updated 2026-06-21 20:00:54 UTC by John