Appendix A: Concept Index by Chapter

The following index maps every major technical concept in the novel to the chapter where it is introduced and the context in which Josh encounters it. It is organized in chapter order - not alphabetically - because the concepts build on each other, and the sequence in which Josh learns them is part of the argument.

Use this index as a study guide, a reference for your own system design, or a map for a second reading focused on the technical layer.

Chapter	Concept	What it means in practice
Prologue / Ch. 1	**Inherited systems and institutional knowledge**	A system built by someone who is gone carries their reasoning in comments, naming conventions, and structural decisions. Reading code is reading the thinking of the person who wrote it.
Ch. 1	**Agentic systems: what they are**	An agent doesn't just reply - it takes actions. It calls tools, reads results, and decides what to do next. The output is behavior, not text.
Ch. 1	**Error rate as diagnostic, not verdict**	A 12.4% error rate is information. It tells you something is structural, not random. The goal is not to eliminate it immediately but to understand what it is.
Ch. 1–2	**Prompt instructions vs. enforced constraints**	A rule in a system prompt is a request. A rule in code is a fact. 'Always verify the account number' and an interceptor that blocks mismatches are not the same thing.
Ch. 2	**Few-shot examples for consistency**	Describing correct behavior produces inconsistent results. Showing the model what correct behavior looks like - with concrete input/output pairs - produces better generalization.
Ch. 3	**The agentic loop**	The loop: send message → receive response → if stop_reason is tool_use, execute tool and append result → send again. The loop ends when stop_reason is end_turn. Your code runs the loop; Claude participates in it.
Ch. 3	**stop_reason: tool_use vs. end_turn**	stop_reason is how your code knows whether to continue the loop or stop. Checking for the string 'tool_use' in raw text is unreliable. Check the API response field.

Appendix A: Concept Index by Chapter

Chapter	Concept	What it means in practice
Ch. 3	**Context accumulation and attention weight**	Every iteration appends to the conversation history. Earlier content receives less attention in longer conversations. A system prompt instruction at the top of turn 1 competes with everything that accumulates after it.
Ch. 4	**Implementing the loop correctly**	The tool execution layer receives a tool call, runs the function, and appends both the call and the result to history. History management and loop termination are separate responsibilities.
Ch. 4	**The track model**	You are not telling Claude what to do. You are building a track it runs on. The control logic lives in your code, not in the model's compliance with instructions.
Ch. 5	**Multi-concern ticket decomposition**	A ticket with multiple issues should be decomposed into parallel workstreams, each with its own context. Parallel threads share relevant customer data but not reasoning. A synthesis step combines results.
Ch. 6	**Business rules in prompts vs. hooks**	A prompt instruction asking the model not to process refunds above $500 is probabilistic. A hook that intercepts and blocks the tool call before it executes is deterministic.
Ch. 7	**Pre-call vs. post-call hooks**	Pre-call interception: enforce rules before a tool runs. Post-call processing (PostToolUse): normalize, filter, and structure tool results before they enter the conversation history.
Ch. 7	**Deterministic vs. probabilistic compliance**	Probabilistic compliance: the model follows the rule most of the time. Deterministic compliance: the rule cannot be violated because it is enforced in code. Speed bumps, not signs.
Ch. 7	**Handoff summaries for human escalation**	When the agent escalates to a human, it should provide a structured summary of the case - not raw transcript. The human reviewer's job is to make a decision, not to reconstruct context.
Ch. 8–9	**Hub-and-spoke multi-agent architecture**	A coordinator receives complex cases and delegates to specialist subagents. The coordinator's job is routing, not reasoning. Subagents are purpose-built and operate independently. A synthesis step combines results.
Ch. 8–9	**Dumb coordinator, smart subagents**	The coordinator should be simple - a dispatcher. Complexity and domain knowledge belong in the subagents. Adding intelligence to the coordinator produces complex failure modes.

Appendix A: Concept Index by Chapter

Chapter	Concept	What it means in practice
Ch. 9	**Parallel subagent execution**	Subagents that don't depend on each other's results can run simultaneously. Sequential execution adds latency with no benefit when there are no dependencies.
Ch. 10–11	**Tool descriptions as the selection interface**	The model selects tools by reading their descriptions. A one-sentence description that overlaps with three others produces misdirection. The description is the only part of the tool the model can see.
Ch. 11	**Negative space in descriptions**	Every tool with a plausible neighbor needs a sentence explaining what it does not handle. Boundary cases are as important as use cases.
Ch. 12	**Tool proliferation degrades reliability**	More tools means more selection surface area. Descriptions that are distinguishable in a set of six become ambiguous in a set of eighteen. Scoped tool access improves both selection accuracy and maintenance overhead.
Ch. 12	**Maintenance contracts**	Every tool is an ongoing commitment: keep it working, keep its description accurate, keep it aligned with the backend. More tools means more implicit commitments.
Ch. 13–14	**Structured error taxonomy**	Uniform error messages prevent appropriate recovery. Four error types - transient, validation, permission, business - warrant four different responses. isRetryable as a boolean makes the retry decision automatic rather than inferred.
Ch. 14	**Consistent failure on a single entity**	If the same tool call fails repeatedly on the same entity, it is probably not transient. A failure counter that steps up classification after three failures prevents indefinite retry loops on data integrity problems.
Ch. 15	**Project vs. user-level configuration**	.mcp.json in the project root is shared infrastructure - version controlled, affecting everyone. ~/.claude.json is personal. Experiments go in the personal config. Proposals for shared changes go through review.
Ch. 16	**tool_choice: auto, any, forced**	Three distinct controls. Auto: the model decides whether to call a tool and which. Any: a tool call is required, the model chooses which. Forced: this specific tool runs now, as the model's next action. Forced selection is not a stronger version of auto - it is a different kind of control.

Appendix A: Concept Index by Chapter

Chapter	Concept	What it means in practice
Ch. 16	**Guaranteed prerequisite steps**	When a step must always occur before any other action, use forced tool selection to make it structurally impossible to skip. Verification before account actions is not a prompt instruction problem. It is a tool_choice problem.
Ch. 18–19	**CLAUDE.md: configuring the tool for the team**	CLAUDE.md is a project-level configuration file that Claude Code reads at session start. It provides the context the tool needs to generate code aligned with the team's conventions - not correct-in-isolation code that requires correction.
Ch. 19	**What to include vs. describe in CLAUDE.md**	State principles, not catalogs. 'Components receive data; they do not fetch it' is more durable than a list of all the places fetching should not occur. The prohibition section is as important as the convention section.
Ch. 21	**Path-specific rules with glob patterns**	Rules that load for every file regardless of context dilute each other. .claude/rules/ files with YAML frontmatter specify which file paths trigger which rules. Testing conventions load for test files; API conventions load for API handlers.
Ch. 22	**Project-scoped vs. user-scoped slash commands**	.claude/commands/ in the repository is version-controlled and shared with everyone who clones. ~/.claude/commands/ is personal. A shared team command belongs in the project. A personal workflow experiment belongs in the home directory.
Ch. 23–24	**Plan mode for complex changes**	Direct execution on multi-file architectural changes produces cascading problems that are expensive to undo. Plan mode surfaces coupling, timing dependencies, and shared state before any file is modified. The plan can be deviated from. It should be read first.
Ch. 25	**Non-interactive CI mode**	Claude Code in CI requires the -p flag (--print). Without it, the session waits for input that will never arrive. With it, it makes reasonable assumptions and documents them rather than blocking.
Ch. 25	**Providing existing tests as context for generation**	Without existing test files in context, test generation produces duplicates. With them, it extends coverage rather than reproducing it.
Ch. 26–27	**Description vs. demonstration in prompts**	Describing an output format produces inconsistent results on cases the description didn't anticipate. Showing the model what correct output looks like - source document and expected

Appendix A: Concept Index by Chapter

Chapter	Concept	What it means in practice
		extraction as a pair - generalizes to novel inputs. Show don't tell is not a literary principle here; it is a reliability principle.
Ch. 27	**Few-shot examples for edge cases**	Four targeted examples each demonstrating a distinct pattern outperform fifteen examples demonstrating the same clean case. Examples covering boundary conditions generalize; examples covering the easy cases don't.
Ch. 28	**Required vs. nullable fields in schemas**	A schema that marks absent fields as required produces fabricated values. The model fills the field rather than returning empty. Null is not empty - null is a finding. Mark fields nullable when they may genuinely be absent.
Ch. 28	**The other enum category**	Every classification schema should include a catch-all category for inputs outside the known taxonomy. A system that cannot say 'I don't know what this is' will say something else instead.
Ch. 28	**Extraction provenance**	Distinguishing direct (verbatim from source), calculated (derived from source values), and absent (not found in source) makes the pipeline's confidence visible and traceable.
Ch. 29	**Retry loops with error feedback**	When schema validation fails, feeding the error back to the extraction step allows the model to correct format mismatches. This works for format errors. It does not work for absent information - the document doesn't become more complete on retry.
Ch. 29	**Full re-validation after correction**	After a successful retry, re-run the full validation pass, not just the previously failed field. A correction to one field can change the validity state of adjacent fields.
Ch. 30	**False positive rates undermine trust**	A flag rate high enough that reviewers learn to dismiss it has failed regardless of detection accuracy. Precision of criteria determines whether a flag produces review or habit.
Ch. 30	**Explicit criteria over vague instructions**	'Check that values are consistent' produces broad flags. 'Flag when the same named field appears with values differing by more than 0.5% between the executive summary and the appendix' produces actionable ones.
Ch. 31	**Latency tolerance as a design requirement**	Batch processing is appropriate when results are not needed synchronously. Pre-submission validation is not batch. Establish latency tolerance

Appendix A: Concept Index by Chapter

Chapter	Concept	What it means in practice
		before architecture - changing it afterward is expensive.
Ch. 31	**custom_id for correlation and lineage**	A well-designed custom_id (customer_id + document_type + timestamp) prevents collision and embeds data lineage in the identifier. The traceability comes free.
Ch. 32	**Independent review instances**	The model that produced an output retains its reasoning. It will not catch its own errors as effectively as an instance reviewing a stranger's work. Two instances see the same output from different positions.
Ch. 32	**Conservative over confident review**	A review instance that over-flags and routes unnecessary items to human review is preferable to one that under-flags and passes errors. In high-consequence domains, the cost of false negatives exceeds the cost of false positives.
Ch. 34–36	**Context degradation vs. failure**	A system that breaks announces itself. A system that drifts does not. Context degradation produces outputs that are subtly wrong - coherent, formatted, plausible, and incorrect in ways that only appear when checked against source material.
Ch. 36	**Verbose tool outputs accumulate**	Tool results appended in full on every iteration become noise that competes with signal. Trim outputs to the fields actually referenced in the conversation before appending them to history.
Ch. 36	**The lost in the middle effect**	Earlier context receives less attention in longer conversations. The customer's original problem statement, stated in message two, may be effectively invisible by message fifteen if it is buried under tool results.
Ch. 36	**Persistent case facts blocks**	A structured block maintained outside the conversation history, populated at intake and updated only at explicit checkpoints, ensures that the information that must always be prominent stays prominent.
Ch. 37	**Claim-source mapping for provenance**	Subagents that produce structured claim-source mappings - verbatim source text paired with the normalized extracted value - prevent the synthesis step from generating figures that have no source. The constraint is structural, not instructional.
Ch. 37	**Conflict annotation over resolution**	When two sources give different values for the same field, the correct output is a conflict

Appendix A: Concept Index by Chapter

Chapter	Concept	What it means in practice
		annotation, not a resolution. The model should surface the conflict; a human should establish the rule for which value takes precedence.
Ch. 38	**Aggregate accuracy masks segment performance**	A 97% accuracy headline may conceal 99% accuracy on eight document types and 80% on three. Test coverage should be stratified by document type, field class, and consequence level - not averaged.
Ch. 38	**Routing thresholds set empirically**	Set routing thresholds against actual segment error rates, not theoretical comfort levels. Low-confidence extractions route to human review. The threshold for what counts as low confidence should come from labeled validation data.
Ch. 38	**The citizen developer model**	AI can do a great deal of what junior engineers do - but only if senior people govern it, configure it, monitor it, and explain it when it goes wrong. The right organizational model is a smaller senior core supporting a broader population of citizen developers, not a headcount reduction.
Ch. 39	**Explicit escalation triggers**	When you want guaranteed behavior, write the condition and the response explicitly. 'Escalate when appropriate' delegates the definition of appropriate to the model. 'Escalate when the customer uses these phrases' does not.
Ch. 39	**Customer preference supersedes model assessment**	When a customer states what they want, believe them. The model's assessment of the optimal outcome is not relevant when the customer has provided their own.
Ch. 40	**Architecture documentation as governance**	An architecture document that records what was built and why is not just reference material - it is the evidence base for governance. Who decided this threshold? On what basis? When the decision-maker is gone, the document is the answer.