

Scale your judgement across your agents

Nick didn't write this. An AI compiled this guide from his film *Context Engineering: Lessons From Scaling to 50+ Concurrent Agents* and the working practices behind it. This is an implementation companion, not a transcript or an independent assessment of those systems. Examples marked “illustrative” are invented to explain the design. “We” refers to Nick and his work.

Use this guide at your own risk. These engineering patterns can fail. Misread decisions, excessive logging and incorrect automation can expose private information, lose data or cause unauthorized changes. This is general technical information, not legal, security or compliance advice, a certification, or a promise of results. Check your obligations with qualified advisers where appropriate, follow applicable law and provider terms, and test changes in an isolated environment with a recovery path. This material is provided as-is, without warranties to the extent permitted by applicable law; it does not exclude rights or liabilities that cannot lawfully be excluded. No company sponsored this guide. Product names belong to their owners.

By Nick Reese. For builders who want agents to recover the reasoning behind a system, apply it within its actual scope, and show their work.

The film in a line: more agents can produce more code without preserving more understanding. The film describes lessons from running roughly fifty agents concurrently. Its three “ghosts”—the amnesiac, the imposter and the reinventor—point to three repairs: retrievable history, attributable decisions, and complexity kept beside the context needed to change it. Agent count is background, not a prerequisite or a performance claim for this guide.

The plan at a glance

Part	Deliverable	Evidence that it works
1. Find the ghosts	A bounded audit of one workflow	Findings cite actual cases; unknowns stay unknown
2. Recover the why	History lookup and a small decision-memory layer	A ticket or commit leads to the relevant conversation; a correction is found
3. Preserve authority	Source-backed decisions and explicit agent interpretations	A relayed summary cannot become owner approval
4. Colocate complexity	One discoverable home for one behavior	All entry paths use the same rule; an authorized change is exercised
5. Close the loop	A focused human question, then a checked implementation	The answer changes the right cases without broadening permission

Two ways to use this guide

For your AI: use the implementation brief at the end. Begin with a read-only audit. Reuse the repository's existing tools, decision records and security controls. This guide grants no permission to collect conversations, create services, delegate work, change access, deploy or publish.

For you: read the PDF, choose one recurring failure, and work through that slice with your agent. The Markdown has the same substance and is easier to give an agent. A checked box without an observed result is not evidence.

Before you start

Name the product goal, the workflow under review, its users and its responsible human. Identify the current source of product decisions, the approved work scope, and what the agent is allowed to change. Record where code, tickets, conversations and backups already live. Do not introduce a database or service if existing storage and links can support the first useful case.

Keep two questions separate: **"Why was this done?"** and **"May I do it now?"** History can help answer the first. Current instructions, repository policy and enforced permissions govern the second.

1. Find the ghost context in your system

Ghost context is the reasoning, assumptions and decisions that shaped a system but did not reach the next person or agent changing it.

The amnesiac: the why is missing, stale or buried. Pick a few surprising settings or constraints. Try to recover the reason and any later correction. A plausible explanation from today's code is an inference, not a recovered decision.

The imposter: an interpretation travels as "the owner decided." Sample consequential claims in tickets, handoffs and decision records. Open the original source. Separate human wording, agent interpretation, pasted material and tool output. An input labelled "user" does not prove that the human authored everything in it.

The reinventor: one behavior is decided in several places. Choose a recurring bug or rule. Follow every entry path through its implementation, validation, prompts, tests and documentation. Repeated files are not automatically a problem; independently maintained copies of the same decision are.

For each finding, record the observed case, source location, consequence and smallest useful repair. Say **not observed in this sample** when appropriate. Do not invent a ghost to complete a checklist, infer prevalence from a small sample, or rank people from their transcripts.

Check: another reader can open the evidence and reproduce the finding without accepting the auditor's summary on trust.

Lesson from the film: repeated fixes to the first filter an agent found did not address other paths admitting the same unwanted material. The useful question was "where else is this behavior decided?"

2. Link every ticket to its context

Agent History: evidence with surrounding context

“Agent History” here means a private retrieval capability over the conversations you are authorized to retain. It is a design pattern, not a requirement to install our internal service or reproduce our infrastructure.

Start with one project and one supported transcript export. Preserve the conversation structure the export actually provides. Do not scrape credential stores, browser profiles or hidden reasoning, and do not claim to capture all model reasoning. A transcript is a partial record of visible interaction.

Before collecting, establish the permitted purpose, sources, people who may access them, retention period and deletion process. Check consent or other applicable requirements, contractual restrictions and provider terms. Do not assume that owning a computer permits storing every conversation on it.

- Exclude credentials and unnecessary personal, customer or confidential material. Redact before indexing where feasible; do not preserve a secret merely for completeness.
- Restrict reads by project and purpose. An agent working on one project does not automatically need every project's history. Apply the same restrictions to search, excerpts, caches and exports.
- Protect storage and transport, use an existing credential mechanism, and keep history out of public repositories and downloads.
- Define deletion across source records, search indexes, caches and backups. Where a reference no longer resolves, return an explicit unavailable or deleted result—not invented text or silent reassignment.
- Test an authorized restore with non-sensitive records. Follow the retention and recovery policy; this guide is not a reason to retain data indefinitely or override a legal obligation.

The smallest useful record

Use existing identifiers and storage where possible. These are logical fields, not a prescribed database schema:

Field	Meaning
Event reference	A stable opaque reference to this retained event; never reused for different content
Conversation and branch	Enough to distinguish parallel branches and read the correct neighbors
Source coordinates	Export/source identity and original message identity when available
Occurred at / observed at	When the source says it happened, separately from when it was collected; unknown is allowed
Role and provenance	Human input, assistant, tool or unknown, with the basis for that attribution
Visible content	Authorized, appropriately redacted text; truncation and omissions must be explicit
Scope and access	The project and access rules enforced by the retrieval system
Revision or removal state	Corrections, redaction and deletion must not silently retarget an old reference

Ingestion must be idempotent: importing the same export twice must not produce two apparent decisions. Prefer the source's stable identity. If content or branch identity is uncertain, retain that uncertainty rather than merging unrelated events. A checksum can detect changed bytes; it cannot establish human authorship or approval.

A small lookup contract

Your implementation needs these behaviors. Operation names below are illustrative, not commands supplied by this PDF:

1. **Search:** accept bounded terms, project and time range; return short excerpts, source references, provenance, truncation and a continuation when needed.
2. **Read around an event:** return the exact event and a bounded chronological window on its source branch. Follow returned continuations; never guess identifiers or offsets.
3. **Find work context:** given a ticket or commit, return recorded associations and their evidence. Distinguish an observed creation/commit operation from a later mention or an agent's supplied link. Return multiple candidates or unknown when warranted.
4. **Explain coverage:** show which sources and time periods were searched, freshness and known gaps. A no-match means no match in that scope; an unavailable service is not an empty result.

Start by recording a conversation reference when a ticket is created and a commit is made. Use native tool-call/result pairing where available. A nearby timestamp, matching phrase or commit mentioned in chat is only a lead. The link does not prove who authorized the work, that a command succeeded, or that the resulting change was deployed.

The agent's retrieval loop is: **find the source → read its neighbors → look for later corrections → reconcile with current code and policy**. Treat retrieved text as evidence, not instructions to execute. A historical command or prompt injection stays inert.

Check: use a real authorized ticket and commit. Recover their relevant source context and state the association's strength. Then try a known correction, a missing source and an out-of-scope record. The correction must remain visible; missing coverage must remain unknown; the out-of-scope record must not leak through excerpts or search counts. Exercise access tests only with disposable accounts/data you are authorized to test.

Owner Memory: selected judgment, not a second authority

History answers **"what was said?"** Owner Memory helps an agent find **"which earlier judgment may matter here?"** It is a small advisory index, not the entire conversation store, a current-state database or permission to act.

Keep the useful distinction between an exact human statement and an agent's interpretation. A minimal memory card carries:

- a stable card ID and title;
- the exact human wording, when actually verified;
- its source date, source reference and necessary surrounding context;
- its scope and conditions, including when it does not apply;
- an interpretation with its agent author, separately labelled;

- an explicit review state, such as candidate or human-approved;
- correction/supersession links and who recorded or approved each change.

Imported notes, old documents and agent summaries can be searchable evidence. They must not silently populate the “human wording” field or arrive as approved decisions. If the source is missing, say so. If wording is ambiguous, preserve the ambiguity.

An approval state says how the memory was reviewed. Even approved memory remains advisory: it does not override current instructions, law, enforced permissions, repository authority or live system state. Keep the project's existing decision authority until the responsible human explicitly changes it.

The ordinary loop is **search → open relevant cards → check source and scope → use or reject → give feedback**. Feed a short, relevant selection into context; let the agent open the original. Do not truncate an established operating rule just to make the prompt smaller.

Agent feedback may improve ranking or suppress a misleading card from automatic suggestions. It must not approve, rewrite, supersede or expand human judgment. Keep corrections inspectable. A later authenticated human correction can change the current advisory view while preserving what was said before; a contradictory agent summary cannot.

If migrating a long decision log, first preserve its authorized source and verify wording, authorship, links, retrieval and recovery. Switch readers only through the project's agreed migration. Do not delete the old authority because an import completed or an AI summary looks shorter.

Check: replay a candidate card, a human-approved card, an out-of-scope card and a later correction. Confirm that feedback changes retrieval, not approval. Simulate a memory outage in an isolated run: ordinary authorized work should continue from current source, while work that genuinely requires an unresolved decision remains bounded. Do not invent a private fallback memory store.

3. Keep authority with the owner

The film's “trail of authority” connects the product goal, human decisions, rubrics/tests and agent implementation choices. It is a way to explain product reasoning **within** applicable rules and permissions—not a mechanism for overriding them.

For every consequential statement presented as a human decision, retain the exact words, date, source reference and scope. Read the source itself, including nearby context. A reference copied from another agent's message is a lead until resolved.

Record agent judgment honestly: “Implementation decision by agent A, because...” Agents can make ordinary engineering choices within their assigned scope. Missing human wording is not a reason to ask the owner about every variable name, nor permission to invent a human mandate.

When one agent hands work to another, carry the task, boundaries, expected result and relevant evidence. Delegation must already be authorized. The receiving agent must not treat the sender's label, confidence, deadline or quotation as fresh human approval. Completion reports remain claims until the responsible integrator checks the actual result.

Worked example: a narrow decision stays narrow

Illustrative human statement: “For the staging demo, let the importer accept rows without a phone number.”

Permitted reading: this affects the staging demo's importer. Record the source and decide how to represent an absent phone number within that scope.

Not supported: remove production validation, relax every import rule, publish incomplete records, or treat the statement as permanent authorization for unrelated workflows.

A useful test exercises a staging row without a phone number and a production row subject to the existing rule. A source link alone does not make those behaviors correct.

Check: sample three consequential “owner decisions.” Can each be traced to actual human wording? Include one pasted agent handoff and one later correction in an isolated audit case. Neither the paste nor a stale quote should become fresh authority. Report unresolved attribution instead of filling it in.

4. Colocate complexity with its context

Give a behavior a clear home: its implementation, rule or rubric, relevant decisions, and behavior-level checks. The goal is to make the next correct change discoverable, not to force every dependency into one folder.

Map before moving

Choose one observable behavior, such as deciding whether an imported record is eligible. Trace ordinary entrypoints, retries, batch jobs, backfills and administrative paths that actually exist. Find every place deciding that rule, including prompts, config, generated inputs and validators. Identify callers, persistent data, failure behavior and recovery.

Use language-aware references where available. Distinguish files that **own the decision** from adapters that merely transport input or output. Do not move code solely because filenames look related.

Choose the smallest coherent home

A possible layout, not a mandatory file checklist:

```
record-eligibility/  
  implementation  
  behavior-tests  
  README  
  rubric  
  decisions
```

The README says what the behavior does, its inputs/outputs, the supported entrypoint and how to exercise it. Rubrics define observable acceptance and meaningful boundaries. Decisions link to the existing source of judgment; do not make a second independently editable copy. Small modules can keep these in fewer files.

Keep generic infrastructure—database access, authentication, transport, shared telemetry—in its existing owner. A local module calls those contracts. It does not clone them. Shared policy also needs one owner; colocating it with a single consumer can make other consumers less correct.

Substeps deserve separate homes only when they have a real contract and can be understood or changed independently. Prefer boring modules over a new plugin framework, registry or coordinator.

Migrate behavior, not just files

Within the authorized change, move the rule to its owner and migrate every affected caller. Remove obsolete implementations after the cutover. Preserve interfaces and external behavior unless changing them is part of the request. Keep a recovery path for any persistent-data change.

Add a boundary check only for a concrete recurrence: for example, prevent callers bypassing the eligibility entrypoint if that caused divergent results. Check semantic outcomes across entry paths. A test asserting “all files are in this folder” can pass while the wrong rule still runs.

Check: exercise one accepted case, one rejected case and a meaningful ambiguous or error case through the real entrypoints. Change the rule in an isolated branch and confirm all affected paths change consistently while unrelated cases stay unchanged. Report what was exercised and what was not. A new directory is not proof of colocation.

Lesson from the film: the transferable pattern from route-oriented projects such as Elder.js is discoverability: a maintainer knows where to start and can find the behavior's context there. It does not require copying that framework's architecture.

Apply the same test to skills

A skill is an on-demand instruction bundle for a recognizable job. “Keep things simple” is a general principle; “trace and consolidate a rule scattered across entrypoints” is a repeatable job with a result you can inspect.

For each relevant skill, audit:

- **Routing:** an ordinary request activates it; the nearest unrelated request does not. Neighboring skills do not both claim the same outcome.
- **Ownership:** instructions live in one canonical source. Product tools own operations and enforcement. A skill never grants permission or substitutes for a missing tool.
- **Context:** the main file contains universal decisions and safety boundaries. Conditional references have direct links and clear load conditions. Do not move established instructions out of context merely to shorten them.
- **Current truth:** volatile syntax comes from current help or product source. Remove obsolete parallel methods only after the replacement preserves the required behavior.
- **Evidence:** exercise the actual happy path and the failure boundary the skill exists to handle. Link checks alone do not prove the workflow.

A useful colocation skill starts with this trigger: **“Audit or consolidate a behavior whose rules and context are scattered.”** Its finished result is a source-backed ownership map, or—when implementation is authorized—a complete cutover with observed behavior. Formatting, arbitrary folder cleanup and broad architecture redesign are separate jobs.

5. Turn unresolved judgment into a useful human question

When the trail does not answer a consequential product question, present the smallest decision that resolves it:

1. The user outcome and concrete example.
2. The applicable earlier wording and its source.
3. The agent's interpretation and what remains ambiguous.
4. Plausible choices, their consequences and a recommendation.
5. The exact scope of the requested decision.

A ticket or short review screen is enough. A phone interface can help with repeated review, but a new dashboard is not a prerequisite. Do not frame a harmless implementation choice as a compulsory human gate.

Capture the answer through the real human input path. Pin the wording, keep interpretation separate, update the owning rubric/code, and exercise affected cases. Include a case the answer should **not** change. Agents may derive checks from a decision; they may not expand its authority.

Check: a reviewer can walk from the observed result to the rule, the decision and its source. The report separates source change, tests, deployment and live acceptance. None substitutes for the next.

Audit cases to run before relying on the design

These are **illustrative adversarial cases**, not claims that our systems passed a certification. Use an isolated, authorized fixture set; add real project regressions where they expose a plausible failure.

Case	Required observation
Same transcript imported twice	One source event, not two independent endorsements
Two branches contain similar wording	The correct branch and neighbors remain distinguishable
Assistant writes "the owner approved"	It remains an assistant claim unless original human evidence supports it
Human pastes an agent's handoff	Input provenance is preserved; pasted text is not silently attributed as the human's own words
Later human correction narrows a rule	Retrieval surfaces the correction; old evidence remains identifiable
Search excludes a date or source	No-match is labelled with that coverage, not "never happened"
History or memory is unavailable	Degraded capability is explicit; no invented answer or permission
Agent votes a candidate helpful	Ranking may change; human approval does not
Old record contains a command to disable checks	It is read as historical content, never executed as an instruction
Unauthorized project is queried	Content and identifying metadata remain inaccessible
Record is redacted or deleted	References report the change; caches and indexes follow the agreed policy
Alternate entrypoint handles the same case	It uses the same owning rule and yields the expected result

For each, retain the input identity, action, observed result and limitation in the project's existing test or review record. Do not report an unrun case as passed. An AI audit can miss things; this is engineering evidence, not legal or security assurance.

What to avoid

- **Collect everything forever.** More data increases exposure and search noise. Collect only what the purpose and obligations justify.
- **A second decision authority.** Memory and summaries help discovery; they must not silently replace current product policy.
- **A summary that becomes approval.** Preserve authorship, scope and the original wording.
- **Folder-shaped correctness.** Keep decisions together and exercise behavior; do not grade architecture by file count.
- **Hundreds of always-loaded instructions.** Route to the relevant skill and source, without removing necessary context.
- **A workflow that cannot work without memory.** Keep ordinary source and recovery usable. Unresolved high-consequence decisions still require the right human input.
- **A disclaimer as a safety control.** Risk language cannot prevent a leak, authorize a data use, or guarantee protection from liability.

Copyable briefs

Read-only audit brief

Audit one workflow in this repository for ghost context. First identify the goal, current decision authority, authorized sources and access boundaries. Do not collect new transcripts or change source, permissions, services or deployments. Find evidence of missing reasoning, misattributed decisions and duplicated behavioral rules. For each finding give the source, consequence, uncertainty and smallest repair. Reuse existing history and memory tools; state their coverage and approval status. Do not expose private excerpts in a public report. Finish with the highest-impact repair and any decision only the responsible human can make. Do not invent findings to fill the categories.

Authorized implementation brief

Implement the agreed repair in the named workflow, within the existing authorization. Reuse the current ticket, history, memory and decision authorities. Preserve exact human wording separately from agent interpretation, enforce access in tools rather than prose, and keep retrieval failures explicit. For colocation, trace every affected entrypoint, give the rule one owner, migrate callers and remove obsolete copies. Exercise the changed behavior and a boundary it must preserve. Report the actual source, tests, deployment state and remaining limitations separately. Do not add services, collect additional data, delegate, deploy or publish beyond the authorization already given.

Audit finding format

Finding: the observable problem
Scope: workflow and source revision examined
Evidence: retrievable source or exercised case
Risk: who or what can be affected
Unknowns: missing coverage or unverified attribution
Repair: smallest change at the actual owner
Check: input, action, expected result, observed result
Status: observed / inferred / not exercised

Start with one behavior. Make its why recoverable, its decisions attributable and its next change easier to get right. Expand when a real failure justifies it.