INTERTECH ENGINEERING CONVERSATIONS

AI Engineering Transition Handbook

Episode 6: Stop Teaching AI the Same Things Over and Over

Purpose: Help IT and engineering leaders identify important knowledge that repeatedly has to be explained, preserve the reasoning that changes engineering decisions, and make that context reusable by developers, assistants, and agents.

Handbook design: Module 6 continues the same cumulative format as Episodes 1-5: episode summary, leadership action checklist, practical examples, working session, worksheets, retained artifacts, tomorrow-morning action, and maturity checkpoint.

Core Idea

Do not document everything. Capture the context that prevents the next engineer or AI from making the wrong reasonable decision. Preserve the reason behind important decisions, give the knowledge an authoritative home, make it reachable from the engineering workflow, and assign ownership for keeping it accurate.

Episode 6 Deliverable

A first Reusable Engineering Context Library for one application, beginning with three to five high-value knowledge entries and a repeatable method for discovering, owning, retrieving, updating, and retiring context.

Episode Summary

Episode 5 focused on engineering constraints that protect maintainability. Episode 6 is broader: it addresses the knowledge that explains how and why a system works the way it does. Source code can reveal much of what exists, but it may not explain why an unusual design remains, why a

business exception matters, or why a previously attempted approach failed.

The goal is not a massive documentation program or an indiscriminate upload of every company document into an AI system. The goal is to identify knowledge that repeatedly changes engineering decisions, preserve it in a trustworthy form, and make it available when the work is performed.

The Episode 6 Context Cycle

1. DISCOVER	2. CAPTURE	3. DELIVER	4. MAINTAIN
Find repeated explanations. Look for what experienced engineers repeatedly have to explain before someone can safely change the system.	**Preserve the reason.** Capture enough context to change the decision, not a history of everything the team knows.	**Make context reachable.** Give it an authoritative home and a practical path into the engineering workflow.	**Own and update it.** Change or retire context when the system, business rule, or underlying reason changes.

A useful knowledge entry answers a practical question: What would a competent engineer or AI reasonably misunderstand if they only had the code and the current task?

IT Leader Action Checklist

Begin with one application or system. The objective is to create a small, useful library before considering broader tooling or enterprise-wide knowledge initiatives.

Find high-value knowledge

- Ask experienced engineers: What do you repeatedly have to explain before someone can safely change this system?
- Listen for phrases such as: 'Before you touch that...' or 'I know this looks strange, but...' These often reveal hidden reasoning.
- Review recurring questions from newer developers, code reviews, support discussions, planning sessions, and repeated AI prompts.
- Prioritize knowledge that changes an engineering decision rather than information that can be readily discovered from the code or system.

Capture the reason, not just the rule

- State what the engineer or AI needs to know.
- Explain why the condition, exception, limitation, or decision exists.
- State what must be understood or verified before changing it.
- Link to a deeper authoritative source when more detail is needed instead of duplicating the full content.

Establish an authoritative home

- Choose one authoritative version for each important piece of context whenever possible.
- Avoid maintaining conflicting copies in repository notes, wikis, prompt files, and other locations.
- Allow other tools or instructions to point to or derive from the authoritative source.

- Record where the authoritative source lives and which application, component, or business capability it applies to.

Make the context reach the work

- Determine how a developer can find the context while working on the affected system.
- Determine how assistants or agents receive the relevant context before making consequential decisions.
- Prefer relevant, scoped context over giving AI indiscriminate access to large amounts of outdated or unrelated material.
- Test retrieval using a real engineering task rather than assuming that stored information will automatically be found.

Assign ownership and lifecycle

- Name the team, role, or person responsible for the accuracy of each important entry.
- Tie updates to changes in the application, architecture, integration, or business rule rather than relying only on calendar reviews.
- Retire knowledge when the reason behind it no longer applies.
- Do not let documented history become permanent policy merely because it was written down.

Measure usefulness

- Watch whether senior engineers have to repeat the same explanation less often.
- Check whether developers and AI make fewer decisions based on missing context.
- If a captured entry is accurate but still not helping, investigate retrieval before creating more documentation.
- Remove, consolidate, or revise entries that are outdated, duplicated, unclear, or no longer affect decisions.

Leadership Self-Check

Question	Yes	Partly	No	Evidence / Notes
Can we identify 3-5 pieces of knowledge that repeatedly have to be explained for this application?	▪	▪	▪	
Does each entry preserve the reason or decision context rather than merely restating code?	▪	▪	▪	
Is there one authoritative home or clearly identified source for each entry?	▪	▪	▪	
Can developers and AI retrieve the relevant context during real work?	▪	▪	▪	
Does each important entry have an owner and a clear update/retirement trigger?	▪	▪	▪	
Are repeated explanations and context-related mistakes decreasing?	▪	▪	▪	

What Knowledge Is Worth Capturing?

Use these categories as discovery prompts, not as a requirement to document everything.

Decision history that changes the obvious solution

- Why two apparently similar processing paths are intentionally separate.
- Why an unusual architecture or integration choice was made.
- What previously attempted approach failed and the condition that caused the failure.

Business context hidden from the code

- A contractual, operational, or regulatory condition that changes how software must behave.
- A customer or product exception that would otherwise look like accidental inconsistency.
- A downstream process or external dependency that is not apparent from the local code.

Operational and system knowledge

- Known system behaviors that matter during deployment, recovery, migration, or support.
- Dependencies whose failure or timing characteristics change the safe implementation.
- Areas where a seemingly local change has consequences elsewhere in the system.

Usually lower priority for this library

- Information AI or another engineer can readily infer from current source code, schemas, or APIs.
- Generic programming knowledge and broad technology documentation available from authoritative external sources.
- Personal preferences that do not materially change correctness, maintainability, risk, or business behavior.
- Old explanations whose underlying constraint no longer exists.

The Capture Filter

Ask	Why it matters
Could a competent engineer or AI discover this reliably from the system?	If yes, another knowledge entry may simply duplicate discoverable information.
Would knowing this change the engineering decision?	Prioritize context that changes what a reasonable engineer would do.
Is the information authoritative and current?	Incorrect context can be more dangerous than missing context.
Can it reach the work when needed?	Stored knowledge without retrieval does not solve the repeated-explanation problem.

Reusable Engineering Context Library

Create the first library with three to five entries. Expand only when real work reveals additional high-value context.

Context / title	Why it matters	Authoritative source	How it reaches work	Owner	Update / retire trigger

Library entry rule

Capture enough context to prevent the wrong reasonable decision - not everything anyone has ever known about the system.

Individual Context Entry Worksheet

Use one worksheet for each high-value item selected for the library.

Context title / system area	
What does someone repeatedly need explained?	
What could a competent engineer or AI reasonably misunderstand without this context?	
What is the important fact, limitation, exception, or decision?	
Why does it exist?	
What must be understood or verified before changing it?	
What can be discovered from the code/system and therefore does NOT need repeating?	
Authoritative source / location	
How will a developer retrieve it during work?	
How will an AI assistant or agent receive it when relevant?	
Owner	
What system/business change should trigger an update?	
What condition should cause this entry to be retired?	
Deeper source or supporting reference, if needed	

Experienced Engineer Knowledge Interview

This is not a request to 'document everything you know.' Use a real application and ask for knowledge that repeatedly changes decisions.

Suggested interview prompts

- What do people repeatedly ask you about this application?
- Where do you usually say, 'Before you change that, you need to know...'?
- What looks wrong or unnecessarily complicated in the code but exists for a real reason?
- What approach has the team already tried that somebody may reasonably try again?
- Which business or operational rule would not be obvious from the repository?
- What change could appear local but actually affect another system or process?
- If you moved to another project tomorrow, what three things would you want the next engineer to know?

Interview triage

Candidate knowledge	Changes a decision?	Discoverable elsewhere?	Capture now?	Owner / source

Retrieval & Trust Test

A knowledge library is only useful if the correct context reaches the work and can be trusted. Test it using a real or representative engineering task.

Test scenario

Engineering task being attempted	
Relevant context entry expected	
Can the developer find it without asking the original expert?	
Does the AI receive or retrieve the correct entry?	
Does the entry contain enough 'why' to change the decision?	
Is any conflicting or outdated context also returned?	
Does the authoritative source match the reusable entry?	
What retrieval or content improvement is needed?	
Retest result	

Trust rules

- Do not equate access to more documents with better context.
- Prefer current, authoritative, task-relevant information over large undifferentiated collections.

- If two sources conflict, resolve ownership rather than asking AI to guess which one is correct.
- Treat retrieval failures separately from content failures: the knowledge may be good but unavailable at the moment it matters.

Context and the Three AI Authority Models

Episode 6 uses the same three models established in Episode 4. Better context can improve all three, but greater authority increases the importance of delivering the right context before action.

Model	Context implication	Leader check
Model 1 - Assistant Human acts; AI helps.	The developer can recognize missing context, pause, and ask an experienced engineer. Reusable context reduces repeated explanation and improves the quality of AI assistance.	Can the developer and assistant find the important context without depending on the same expert every time?
Model 2 - Agent with Human Approval AI acts; human approves.	The agent may make several decisions before review. Relevant system and business context should be available before execution, with human approval retained for judgments that remain uncertain.	Does the reviewer receive enough context and evidence to recognize whether the agent respected the reason behind the system?
Model 3 - Bounded Autonomous Agent AI acts within limits.	The workflow cannot depend on a human remembering to explain history halfway through the task. Required context must be reliably available within the agent's bounded work environment.	Is the required context current, authoritative, retrievable, and sufficient for the delegated decision? If not, reduce authority or add approval.

Working Session: Build the First Context Library

Recommended format: 60-90 minutes with the engineering leader, two or three people who work with the selected application, and at least one experienced engineer who knows its history.

Opening prompt:

"What do we repeatedly have to explain before another engineer - or AI - can safely change this system?"

Working sequence

- Choose one application and collect repeated explanations from the team.
- Select three to five candidates where missing context could produce a wrong but reasonable decision.
- Complete an Individual Context Entry Worksheet for each selected item.
- Identify the authoritative source and eliminate or flag conflicting copies.
- Decide how developers, assistants, and agents will receive the context during actual work.
- Assign ownership and define the system or business event that should trigger an update or retirement.
- Test at least one entry against a real engineering task and record retrieval problems separately from content problems.

Working-session outputs

The session should end with a first Reusable Engineering Context Library, three to five completed entries, named owners, retrieval paths, update/ retirement triggers, and at least one tested context-to-workflow path.

Working notes

Tomorrow-Morning Action & Handbook Retention

Tomorrow morning:

- Choose one application.
- Bring together two or three people who know it, including at least one experienced engineer.
- Ask: What do you repeatedly have to explain before someone can safely change this system?
- Select **three to five** high-value answers.
- For each one, capture the important context, the reason behind it, and what must be understood before changing it.
- Give each entry an authoritative home, a path into the engineering workflow, an owner, and an update or retirement trigger.
- Watch whether the repeated explanation begins to disappear.

What this step accomplishes

The organization begins converting institutional memory into reusable engineering context. Senior engineers spend less time repeating history, developers and AI receive better context earlier, and important reasoning survives personnel or project changes without turning old decisions into permanent rules.

Retain for the cumulative handbook

Module	Focus	Status	Key retained artifact
Episode 1	Make AI usage visible	Complete	AI Usage Inventory + Learning Loop
Episode 2	Validate lifecycle productivity	Complete	Lifecycle Evidence + Reliable Leverage Candidates
Episode 3	Create shared engineering standards	Complete	First Standards + Classification + Improvement Path
Episode 4	Set appropriate AI authority	Complete	AI Autonomy Map + Agent Boundaries
Episode 5	Prevent AI-generated technical debt	Complete	AI Technical Debt Control Map
Episode 6	Capture reusable engineering knowledge	Complete	Reusable Engineering Context Library

- **Reusable Engineering Context Library** - the authoritative index of high-value context for the selected application.
- **Individual Context Entry Worksheets** - the reason, risk of misunderstanding, retrieval path, owner, and lifecycle for each entry.
- **Experienced Engineer Interview/Triage Sheets** - evidence of where repeated explanation exists and which knowledge was prioritized.
- **Retrieval & Trust Tests** - proof that relevant context can reach developers and AI during actual work.
- **Ownership and Lifecycle Record** - who maintains each entry and what triggers update or retirement.

Episode 6 Maturity Checkpoint

- We can identify important knowledge that repeatedly has to be explained.
- We capture reasoning that changes decisions rather than duplicating information AI can readily discover.
- Each high-value entry has an authoritative source and owner.
- Developers and AI can retrieve relevant context during the engineering workflow.
- Context is updated or retired when the underlying system or reason changes.
- We distinguish a content problem from a retrieval problem.
- Repeated explanations and context-related mistakes are measurably declining.