INTERTECH ENGINEERING CONVERSATIONS

AI Engineering Transition Handbook

Episode 5: Preventing AI-Generated Technical Debt

Purpose: Help IT and engineering leaders prevent increased AI production speed from becoming increased complexity, architectural drift, duplication, and future maintenance cost.

Handbook design: Module 5 continues the same cumulative format as Episodes 1-4: episode summary, leadership action checklist, working session, detailed worksheets, retained artifacts, tomorrow-morning action, and maturity checkpoint.

Core Idea

Do not try to control every line AI generates. Identify the engineering decisions that protect the software, make those decisions clear to AI, and build practical controls around the ones the organization cannot afford to repeatedly get wrong.

Episode 5 Deliverable

An **AI Technical Debt Control Map** for one application or engineering team. It identifies the few engineering constraints that protect maintainability, how AI receives those constraints, how each is validated or reviewed, who owns the control, and how recurring AI mistakes are converted into stronger guidance or controls.

Episode Summary

Episode 2 asked whether AI was producing real lifecycle productivity. Episode 5 addresses a different problem: as AI helps the organization create and change software faster, how does the organization prevent technical debt from being created faster too?

For this handbook, technical debt means a design or implementation choice that makes future engineering work harder or more expensive than it needs to be. AI did not create technical debt, but it changes its potential scale. A weak pattern, unnecessary abstraction, or duplicate implementation can now be generated and repeated very quickly.

The goal is not a separate development process for AI. AI-generated work should meet the same architecture and engineering standards as human-generated work. Where AI exposes a weakness in the existing engineering process, strengthen that process rather than simply telling developers to inspect AI more carefully.

The Episode 5 Control Cycle

1. IDENTIFY	2. MAKE VISIBLE	3. PROTECT	4. LEARN
Choose the few constraints that protect maintainability. Focus on decisions where repeated inconsistency creates meaningful future cost.	**Make the constraint available to people and AI.** Do not rely on AI to infer important architecture from whichever files happen to be in context.	**Automate dependable checks; retain human judgment where needed.** Keep changes small enough for meaningful review.	**Turn meaningful recurring mistakes into better guidance or controls.** Do not repeatedly pay to discover the same failure.

Episode 4 remains important here. Model 1 - Assistant keeps the human acting. Model 2 - Agent with Human Approval allows AI to act but retains human judgment at an important boundary. Model 3 - Bounded Autonomous Agent requires dependable validation for the decisions being delegated. If an important architectural judgment cannot be validated reliably, retain the human approval point.

IT Leader Action Checklist

Start with one application or one engineering team. Do not attempt to solve technical debt control across the entire organization at once.

Identify the constraints that matter

- Select **three to five** engineering decisions where repeated inconsistency would create meaningful future cost.
- Focus on maintainability, architecture, system boundaries, approved patterns, and important dependency decisions - not personal coding preferences.
- Use Episode 3 standards as a starting point, but narrow the list to the constraints that most directly protect this application.
- Write each constraint as a clear engineering expectation that a developer and AI can understand.

Make the constraints available to AI

- Determine how each important rule reaches the AI doing the work: repository instructions, project guidance, reusable context, tool configuration, or another persistent mechanism.
- Do not rely on AI to rediscover an important rule by examining a few nearby source files.
- Keep guidance specific to the system and task. Avoid burying critical constraints inside large documents that are difficult for people or tools to apply.

Decide how each constraint is protected

- Use automated validation when the rule can be checked dependably through tests, builds, analysis, policy checks, or other engineering controls.
- Retain human engineering judgment when the decision cannot be reduced to a dependable automated check.
- For agent work, align the control with Episode 4 authority: if an important judgment cannot be validated reliably, keep the human approval point.
- Confirm that the control exists in the normal engineering path rather than depending on someone remembering to perform a special AI-only review.

Keep AI-generated changes understandable

- Separate unrelated feature work, refactoring, dependency changes, and cleanup when combining them would make review difficult.
- If a reviewer cannot reasonably explain what changed and why, reduce the size or scope of the change.
- Treat understandable change size as a technical-debt control, not merely a review preference.

Look specifically for debt patterns AI can accelerate

- **Duplication:** Did AI add a helper, service, rule, or capability the system already has?
- **Unnecessary complexity:** Did AI introduce abstractions, frameworks, or flexibility that the current problem does not require?
- **Architectural drift:** Does the change solve the immediate requirement while moving responsibilities or patterns away from established system design?
- **Dependency growth:** Did the change introduce a new library, service, or integration when an approved capability already exists?
- **Inconsistent behavior:** Did AI create a second way to handle validation, errors, security, logging, or another cross-cutting concern?

Turn recurring problems into better controls

- Track meaningful AI-generated issues that recur; do not create a rule for every imperfect choice.
- For a repeated problem, determine whether the root cause is missing context, unclear guidance, weak validation, excessive change size, or an approval gap.
- Improve the smallest useful part of the system: guidance, automated control, review point, or agent boundary.
- Recheck whether the same problem occurs less often after the improvement.
- Remove or revise controls that add significant effort without preventing a meaningful problem.

Leadership Self-Check

Question	Yes	Partly	No	Evidence / Notes
Can we name the 3-5 engineering constraints that most protect this application?	▪	▪	▪	
Can the AI used on this system access those constraints without guessing?	▪	▪	▪	
Do we know which constraints are automated and which require human judgment?	▪	▪	▪	
Are AI-generated changes kept small enough for meaningful review?	▪	▪	▪	
Are we identifying meaningful recurring debt patterns rather than isolated preferences?	▪	▪	▪	
Do we improve or remove controls based on evidence that they are working?	▪	▪	▪	

Engineering Constraint & Review Example Library

Use these examples to prompt discussion. They are not universal rules and should not all be adopted. Select only constraints that protect the software you actually have.

Architecture and responsibility boundaries

- Where business logic belongs and which layer or component owns a responsibility.
- How data access is performed and which code is permitted to reach persistence directly.
- How modules or services communicate and which boundaries should not be bypassed.

- Which existing patterns should be extended rather than replaced by a new approach.

Maintainability and reuse

- Search for an existing capability before adding a new helper, service, abstraction, or utility.
- Prefer the simplest implementation that satisfies the current requirement and established design.
- Avoid speculative abstractions created only for possible future requirements.
- Keep shared behavior in the established shared location rather than duplicating it near each feature.

Dependencies and platform consistency

- Use approved libraries and platform capabilities before introducing new dependencies.
- Require a reason for adding a dependency that duplicates an existing capability.
- Respect supported versions, frameworks, runtime choices, and migration direction for the application.

Cross-cutting engineering concerns

- Follow established approaches for authentication, authorization, validation, error handling, logging, and telemetry.
- Do not create feature-specific security or operational patterns when the application already provides them.
- Preserve testing expectations for the type of behavior being changed.

Useful review prompts

Question	Follow-up
Did AI add something the system already had?	If yes, why was reuse not possible?
Did AI make the application more complicated than the problem required?	If yes, what current requirement justifies the complexity?
Does the change follow the system's existing architecture?	If no, is this an intentional architecture decision or accidental drift?
Can the reviewer explain what changed and why?	If no, reduce or split the change before approval.
Would we accept this design if a human engineer had written it?	AI speed should not lower the engineering standard.

AI Technical Debt Control Map

Complete one row for each selected engineering constraint. Keep the first map small enough to be used and maintained.

Engineering constraint	Why it matters	How AI receives it	Protection method	Owner	Evidence it works

Protection Method Key

- **Automated:** a dependable test, build, analysis, policy, or other machine-enforced control.
- **Human judgment:** an engineer must evaluate the design or result at a meaningful review point.
- **Combined:** automated evidence narrows the review while a human retains a consequential decision.

Primary decision prompt

"What are the few engineering decisions we cannot afford to let AI repeatedly get wrong?"

AI-Generated Change Review Worksheet

Use this worksheet when AI produces a substantial change or when reviewers report that generated changes are becoming difficult to understand.

Application / repository	
Feature / change objective	
AI model in use	Model 1 - Assistant | Model 2 - Agent with Human Approval | Model 3 - Bounded Autonomous Agent
What changed?	
Why did it need to change?	
Can a reviewer explain the change without reconstructing the entire task?	
Does the change mix unrelated feature work, refactoring, dependency changes, or cleanup?	
Did AI add a capability the system already had?	
Did AI add abstractions or flexibility not required by the current problem?	
Does the change follow the selected architectural constraints?	
What automated validation passed?	
What still requires human engineering judgment?	
Should this change be split before approval? Why?	
Decision / owner	

Recurring AI Debt -> Control Improvement Worksheet

Use this only for problems that are both **meaningful and repeatable**. The purpose is to stop paying to discover and correct the same failure downstream.

Recurring problem observed	
Where / how often has it appeared?	
Future engineering cost if it continues	
Likely root cause	Missing context | Unclear guidance | Weak validation | Excessive change size | Approval gap | Other
What should AI have known before making the change?	
Could the issue be detected dependably by automation?	
If not, where should human judgment occur?	
Smallest useful improvement	Guidance | Context | Automated check | Review point | Agent boundary | Other
Owner	
Implementation date / review trigger	
Evidence to collect	
After review: is the problem occurring less often?	
Keep, revise, or remove the control?	

Control effectiveness rule

A new rule or validation step is not automatically valuable. If a control adds meaningful engineering effort but does not reduce a meaningful problem, revise or remove it. The control system itself should not become unnecessary technical debt.

Technical Debt Controls by AI Authority Model

Use the same three-model terminology established in Episode 4. Episode 5 does not create a new autonomy model; it applies maintainability controls to the authority already chosen.

Model	Who acts?	Technical-debt implication	Minimum expectation
Model 1 - Assistant	Human acts; AI helps.	The developer remains inside the work and can apply engineering judgment while implementing.	Give AI relevant constraints and require the normal engineering review and validation path.
Model 2 - Agent with Human Approval	AI acts; human approves at an important boundary.	AI may make multiple implementation decisions before the human sees the result.	Provide constraints before execution, collect validation evidence, and place human approval where architectural or maintainability judgment still matters.
Model 3 - Bounded Autonomous Agent	AI acts inside predefined limits.	The organization cannot depend on a developer noticing debt while the work is being created.	Only delegate decisions that are bounded and can be validated dependably. Retain human approval for important judgments that cannot be validated reliably.

Before increasing agent authority

- Confirm the relevant engineering constraints are available to the agent automatically.
- Confirm the agent cannot bypass the validation path that protects those constraints.
- Review representative failures, not only successful demonstrations.
- Verify that recurring debt is not increasing as human involvement decreases.

If a critical maintainability decision still requires expert judgment, keep that judgment in Model 2.

Working Session:
Build the First Technical Debt Control Map

Recommended format: 60-90 minutes with the engineering leader, one or two experienced engineers who know the selected application, and the person responsible for the AI-assisted development workflow. Bring the Episode 3 standards and Episode 4 Autonomy Map.

Opening prompt:

"What are the three to five engineering decisions that protect this application, and how do we make them harder for both AI and humans to violate?"

Working sequence

- Choose one application or team and name three to five high-value maintainability constraints.
- Write each constraint clearly and identify how AI receives it.
- Classify the protection as automated, human judgment, or combined.
- Review recent AI-generated work for duplication, unnecessary complexity, architectural drift, and oversized changes.
- Choose one meaningful recurring problem, if one exists, and complete the Control Improvement Worksheet.
- Assign an owner and define what evidence will show that each new control is helping.
- Set a review trigger rather than allowing controls to accumulate indefinitely.

Working-session outputs

The session should end with a completed first AI Technical Debt Control Map, at least one documented change-review decision, any recurring-

problem improvement selected for implementation, and named owners for the controls.

Working notes

Tomorrow-Morning Action & Handbook Retention

- The podcast assignment and handbook exercise should produce the same result. Tomorrow morning:
- Choose one application or engineering team.
- Identify three to five engineering constraints that protect its long-term maintainability.
- Make each constraint clear and available to the people and AI working on the system.
- Decide whether each constraint can be protected automatically or still requires human engineering judgment.
- Review AI-generated work for meaningful recurring problems and changes too large to understand confidently.
- When a meaningful problem repeats, improve the guidance or control instead of repeatedly fixing the same issue downstream.

What this step accomplishes

When complete, the organization has moved beyond the instruction to 'review AI code carefully.' It has begun creating an engineering environment in which the decisions that protect maintainability are explicit, available to AI, connected to dependable controls, and improved using evidence from real failures.

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	Pending	To be added

- **AI Technical Debt Control Map** - the selected constraints, how AI receives them, protection method, ownership, and effectiveness evidence.
- **AI-Generated Change Review Worksheets** - examples of substantial changes and the engineering judgment applied.
- **Recurring AI Debt -> Control Improvement Worksheets** - repeatable failures and the guidance or control changes created from them.
- **Control Effectiveness Evidence** - whether the targeted debt pattern occurs less often and whether the control is worth its cost.
- **Links to Episode 3 and 4 artifacts** - standards and autonomy decisions that govern the same application.

Episode 5 Maturity Checkpoint

- We can name the few engineering constraints that most protect the selected application.
- Those constraints are available to AI rather than left for it to infer.

- We know which constraints can be validated automatically and which still require human judgment.
- AI-generated changes remain small enough for meaningful review.
- We distinguish meaningful recurring debt from isolated style preferences.
- Recurring AI mistakes feed improvements to guidance, validation, review, or agent boundaries.
- We evaluate whether controls are actually reducing problems and remove controls that do not justify their cost.