INTERTECH ENGINEERING CONVERSATIONS

# AI Engineering Transition Handbook

Episode 2: AI Saves Time, Until It Doesn't

**Purpose:** Help an IT or engineering leader determine whether reported AI productivity gains are true end-to-end engineering improvements or whether work is simply being shifted into review, correction, testing, integration, maintenance, or future technical debt.

## Core idea

Do not measure AI productivity where the AI stops. Measure it where the engineering outcome becomes real. A developer finishing faster is useful; an organization eliminating real work, improving the result, or avoiding expensive mistakes is better.

## Episode 2 Deliverable

A small set of AI-assisted practices traced through the software lifecycle, evidence showing where each practice creates or shifts work, the conditions required for it to succeed, and a short list of practices ready for additional validation before standardization.

# Episode Summary

Episode 1 made current AI usage visible and created a learning loop. Episode 2 takes the next step: it asks whether the apparent productivity gains are real when viewed across the engineering lifecycle rather than only at the developer's keyboard.

The central distinction is between local productivity and lifecycle productivity. Local productivity asks whether AI helped one person complete a task faster. Lifecycle productivity asks whether the organization reduced the total effort required to deliver, support, and later change the result. Time saved during generation can reappear as

extra code review, correction, testing, integration work, architectural cleanup, documentation gaps, or maintenance burden.

Not every high-value AI use has to save time directly. AI may also improve engineering value by helping an engineer discover a hidden dependency, compare plausible options, investigate a failure faster, or avoid a poor design decision.

These benefits often appear as work that never had to occur.

The episode therefore recommends looking for reliable leverage: AI practices that repeatedly remove effort, improve the outcome, or reduce avoidable risk without creating disproportionate downstream work. A spectacular one-time result is useful evidence, but it is not yet a repeatable organizational practice.

## The progression introduced in this episode is:

| # | Stage | Leadership intent |
|---|---|---|
| 1 | **Select** | Choose a few recurring AI practices from the Episode 1 inventory. |
| 2 | **Trace** | Follow the work past the point where the developer says the AI-assisted task is finished. |
| 3 | **Evaluate** | Determine whether the practice removed work, improved the outcome, or reduced risk. |
| 4 | **Identify conditions** | Capture the context, engineering judgment, and validation required for success. |
| 5 | **Repeat** | Test several comparable examples so evidence replaces anecdotes. |
| 6 | **Retain** | Keep the evidence so Episode 3 can decide what is ready to become a shared standard. |

# IT Leader Action Checklist

Use this checklist after completing the Episode 1 discovery exercise. The objective is not to prove that AI works or does not work. The objective is to determine where the benefit survives the full engineering process.

## Select the practices to examine

- Start with the AI Usage Inventory retained from Episode 1.
- Choose three practices that occur often enough to matter. Avoid selecting only the most impressive demonstrations.
- Include at least one practice that creates a permanent engineering artifact, such as production code, tests, configuration, documentation, or infrastructure.
- Include at least one practice that primarily supports understanding or decision making, such as legacy-code analysis, debugging assistance, research, or impact analysis.
- Prefer practices where the team can compare AI-assisted work with similar non-AI work or with prior experience.
- Identify a technical reviewer who understands the affected system well enough to judge architectural fit and downstream consequences.

## Establish the claimed benefit

- Record what the developer believes AI improved before reviewing the downstream work.
- Capture the original claim in plain language: for example, 'reduced discovery from half a day to one hour' or 'generated a useful starting set of tests in ten minutes.'
- Separate the visible benefit from the business assumption. A faster coding step is not yet proof of faster delivery.
- Do not use generated lines of code, prompt count, or AI usage frequency as primary productivity measures.

- Ask whether the value claimed is time saved, a better engineering outcome, reduced risk, or some combination.

### Trace the productivity downstream

- Follow the practice through the next meaningful engineering stages instead of stopping when the AI output is produced.
- Record unusual review effort. Examples include architectural corrections, explanation time, additional senior review, or repeated clarification.
- Record unusual correction or rework. Note whether a developer had to rewrite a significant part of the AI output.
- Record testing effects. AI may reduce test authoring time while creating additional debugging, false confidence, or missing meaningful behavior.
- Record integration effects. Look for unexpected dependencies, duplicated functionality, inconsistent patterns, or changes another team must absorb.
- Where practical, note delayed effects such as maintainability concerns, confusing code, weak documentation, or future cleanup obligations.

## Evaluate total engineering value

- Ask: Did this practice reduce total engineering effort after review, correction, testing, integration, and handoff?
- Ask: Did it improve correctness, understanding, maintainability, or another engineering outcome?
- Ask: Did it reduce the chance or potential cost of a mistake?
- Distinguish work eliminated from work relocated. Moving effort from the developer to a reviewer is not the same as removing it.
- Give credit for avoided work. Discovering a dependency before implementation or preventing a defect can be more valuable than visible time savings.

- Do not require every valuable practice to produce a precise percentage improvement. Early evidence can be directional if the reasoning is documented.

## Capture the conditions for success

- Record what context had to be supplied to the AI for the practice to work well.
- Record what the engineer already needed to know in order to challenge or verify the output.
- Record the validation step that made the result trustworthy.
- Identify whether the practice depends heavily on senior engineering judgment.
- Identify where the same workflow may be inappropriate for a less experienced engineer or a higher-risk system.
- Repeat the practice across several comparable examples before treating it as evidence of reliable leverage.
- Do not standardize the practice yet. Retain the evidence for the standards discussion in Episode 3.

## Leadership self-check

| Question | Yes | Partly | No | Evidence / Notes |
|---|---|---|---|---|
| Can we distinguish local task speed from end-to-end engineering productivity? | ■ | ■ | ■ | |
| Have we traced at least three AI practices beyond the developer's completion point? | ■ | ■ | ■ | |
| Do we know where AI is removing work versus shifting it downstream? | ■ | ■ | ■ | |
| Do we know what context, judgment, and validation make each promising practice succeed? | ■ | ■ | ■ | |
| Do we have repeated evidence rather than one-time productivity stories? | ■ | ■ | ■ | |

# Working Session: Trace the Productivity

**Recommended format:** 60-90 minutes with the technical reviewers and several developers whose Episode 1 examples were selected. Examine three practices. The objective is evidence, not a debate over whether AI is good or bad.

***Opening prompt***

***"Take one AI-assisted practice we believe saves time. Follow it from the developer's first use of AI to the point where the engineering outcome is accepted. Where was work actually removed, where did work reappear, and what had to be true for the result to be trustworthy?"***

## For each practice, ask:

- What was the engineering task, and what did AI contribute?
- What did the developer believe was saved or improved at the moment the AI-assisted work was completed?
- What happened during review, correction, testing, integration, deployment, or handoff?
- Did another engineer spend additional time understanding or correcting the result?
- Did AI create a permanent artifact that the organization now has to own and maintain?
- Did AI help prevent work by finding a dependency, identifying a defect, clarifying an unfamiliar area, or exposing a poor approach early?
- Would another qualified engineer obtain a similar benefit, or is the result highly dependent on one person's expertise?
- What context did the AI need? What did the engineer need to know? How was the result verified?

- Would we be comfortable repeating this practice several times before deciding whether to promote it?

## Examples of hidden downstream work to watch for

- **Review:** extra senior-engineer time, architectural correction, explaining why a generated approach does not fit existing patterns.
- **Correction:** rewriting plausible but unsuitable code, replacing unnecessary dependencies, removing duplicated functionality.
- **Testing:** debugging generated tests, correcting weak assertions, discovering that green tests did not cover the important behavior.
- **Integration:** resolving contract mismatches, unintended coupling, configuration differences, or work pushed to another team.
- **Maintenance:** code that is harder to understand later, inconsistent patterns, unnecessary volume, missing rationale, or hidden future cleanup.

## Examples of value that may not look like time savings

- A hidden dependency is found before implementation begins.
- An engineer compares several approaches and rejects a technically plausible but architecturally poor option.
- A defect or edge case is identified before it reaches production.
- A developer understands an unfamiliar system well enough to ask better questions sooner.

## Session notes

# Lifecycle Productivity Evaluation Worksheet

Complete one copy for each AI-assisted practice selected for deeper evaluation. Use the worksheet to trace the claimed benefit into the full engineering outcome.

| | |
|---|---|
| **Practice / use case** | |
| **Engineering task or problem** | |
| **Developer(s) / reviewer(s)** | |
| **AI tool / model / environment** | |
| **Claimed local productivity gain** | |
| **Expected engineering outcome** | |

| Criterion | Evaluation question | Strong | Mixed | Weak |
|---|---|---|---|---|
| **Total effort** | Did the practice reduce total effort after review, correction, testing, and integration? | ■ | ■ | ■ |
| **Downstream work** | Did meaningful work reappear after the developer considered the task complete? | ■ | ■ | ■ |
| **Quality** | Was the final result at least as correct and useful as comparable work? | ■ | ■ | ■ |
| **Risk reduction** | Did AI help avoid a defect, bad decision, hidden dependency, or other expensive mistake? | ■ | ■ | ■ |
| **Architecture** | Did the result fit established boundaries, patterns, and intentional dependencies? | ■ | ■ | ■ |
| **Maintainability** | Is the result understandable and supportable by another engineer later? | ■ | ■ | ■ |
| **Verification** | Can the result be checked with a clear and proportionate validation step? | ■ | ■ | ■ |
| **Context** | Can the required system, business, and engineering context be supplied consistently? | ■ | ■ | ■ |
| **Judgment** | Is the amount of senior engineering judgment required understood and acceptable? | ■ | ■ | ■ |
| **Repeatability** | Has the practice produced similar benefits across several representative examples? | ■ | ■ | ■ |

## Disposition

- Continue experimenting - promising, but more lifecycle evidence is needed.
- Repeat with comparable work - the benefit appears real but has not yet been demonstrated consistently.

- Candidate for shared practice - evidence shows reliable leverage and acceptable validation cost.
- Refine the workflow - benefit exists, but context, review, or validation needs improvement.
- Do not promote - downstream effort, quality, architecture, maintainability, or risk outweighs the benefit.

## AI Practice Evidence Log

Use this page to summarize repeated observations for one promising practice. The goal is to move from a one-time story to evidence that can support the standards discussion in Episode 3.

| Evidence item | Example 1 | Example 2 | Example 3 |
|---|---|---|---|
| **Claimed local time / effort saved** | | | |
| **Review or correction added later** | | | |
| **Testing / integration effect** | | | |
| **Quality or risk improvement** | | | |
| **Maintainability concern** | | | |
| **Context required** | | | |
| **Validation used** | | | |
| **Engineering judgment required** | | | |
| **Overall result** | | | |

## Conditions required for success

AI context or instructions that materially affect the result:

_____

Human knowledge or experience required:

_____

Required validation or review step:

_____

Known situations where this practice should not be used:

_____

## Evidence conclusion

Reliable leverage demonstrated

More evidence needed

Workflow needs refinement

Do not promote

# What to Retain for the Handbook

Episode 2 should add evidence to the artifacts created in Episode 1. Retain the following so the standards discussion in the next module begins with facts rather than opinions.

- **Lifecycle Productivity Traces** - selected practices followed from initial AI use through the accepted engineering outcome.

- **Hidden Work Findings** - examples of review, correction, testing, integration, handoff, or maintenance effort that reduced the apparent productivity gain.
- **Avoided Work / Risk Findings** - cases where AI prevented defects, exposed dependencies, improved a decision, or avoided unnecessary implementation.
- **Reliable Leverage Candidates** - practices that repeatedly removed effort or improved outcomes without disproportionate downstream cost.
- **Conditions for Success** - required context, engineering expertise, system knowledge, and validation steps.
- **Practice Evidence Logs** - repeated observations showing whether a benefit is reproducible.
- **Practices Not Ready to Promote** - useful experiments that need refinement, stronger evidence, or a different control before wider use.

## Episode 2 maturity checkpoint

- You are ready to move forward when you can reasonably say:
- We can distinguish developer-level speed from lifecycle productivity.
- We have traced several AI-assisted practices through downstream engineering work.
- We are measuring value as effort removed, outcome improved, or risk reduced - not simply AI activity or generated code volume.
- We know the context, validation, and engineering judgment required for our most promising practices.
- We have repeated evidence for at least a few practices rather than relying only on anecdotes.
- We have not yet confused a promising practice with a final engineering standard.