DEVDAY.org Inspiring IT Movement

AXON ACTIVE Focusing on your agility

Workshop Synthesis Report

AI Adoption & Junior Developer Growth

Software Quality, Security, and the Path from Junior to Senior

Based on word cloud responses from multiple development teams across Vietnam at the DevDay 2026 in Da Nang

April 2026 | DevDay 2026

Thank you for participating!

This report synthesizes the word cloud responses collected during the DevDay 2026 workshops. It is intended as a starting point — an input for your team's own discussions, not a final verdict. Your specific context, constraints, and priorities will naturally shape how these findings apply. The collected raw results are attached in the Appendix.

If you have additional feedback, corrections, or insights to add, we'd love to hear from you: **ai@axonactive.com**

Methodology note: All workshop responses were collected anonymously. The analysis was performed using Claude Opus 4.6 in a zero-history session to avoid any prior bias.

The Core Premise

- AI is a gift — it makes us faster, sometimes 50% or more.
- The goal is not to fight AI or slow it down.
- The goal is to use AI right: be more efficient, more secure, and stay in control.
- If juniors never earn the basics — debugging, architecture thinking, security awareness, reading code critically — then in 5 years we have senior titles with junior judgment, and nobody left who can catch what AI gets wrong.

Contents

SECTION A — For Companies and Team Leads

The Core Tension

AI dramatically accelerates code production, but when teams adopt it without guardrails, they trade speed for a growing deficit in critical thinking, code ownership, security awareness, and architectural judgment — the exact skills that separate reliable software from fragile prototypes.

Top 5 Risks to Junior Developer Growth and Software Quality

The following risks emerged directly from the workshop teams' word cloud responses:

#	Risk	What Teams Reported
1	**Reduced Critical Thinking**	Teams repeatedly flagged "reduced critical thinking," "less critical thinking," "skipped deep thinking," and "do more think less." When AI generates code instantly, juniors skip the reasoning process that builds problem-solving ability.
2	**Blind Trust in AI Output**	"Blind trust in AI outputs," "trust AI too much," and "high dependency on AI" appeared prominently. Developers accept AI-generated code without verifying correctness, security, or fit for the specific context.
3	**Loss of Code Ownership**	Teams noted "less ownership of code," "copy paste coding," "AI work not human work," and "no one knows what they do." When developers don't write or deeply understand the code, they cannot maintain, debug, or extend it.
4	**Surface-Level Problem Solving**	Responses like "surface level," "ignoring edge cases," "some quick test," and "blind AI usage" reveal that AI encourages solving only the visible layer of a problem while missing edge cases, error handling, and architectural implications.
5	**Weakened Security and Architecture Judgment**	Teams identified "security," "system architecture," and "business logic" as areas where AI speed comes at the cost of understanding. AI can produce working code that has security holes or architectural flaws invisible to inexperienced developers.

Top 5 Root Causes Behind These Risks

#	Root Cause	Explanation (from workshop data)
1	**Struggle-based learning is disappearing**	Image 1 reveals that developers grew strongest through struggles: “fixing bug at 2am,” “my first big failure,” “struggling with OOP C++,” “overcoming difficulties.” AI removes these productive struggles before juniors can learn from them.
2	**No feedback loop on AI-generated code**	Code review was the single largest word in Image 3 (proposed solutions), indicating teams know reviews are missing or insufficient. Without rigorous review of AI output, errors and bad patterns propagate silently.
3	**Speed is rewarded over understanding**	The workshop question itself — “where does AI increase speed but weaken understanding?” — confirms that teams see velocity as the default metric. Responses like “project with little time” and “fast information” show that deadline pressure pushes teams toward unchecked AI usage.
4	**Juniors lack the baseline to evaluate AI**	Without debugging experience, architecture knowledge, and security awareness, juniors cannot tell good AI output from dangerous AI output. Image 2 shows “lack of accuracy” as a key concern — but accuracy can only be judged by someone who understands the domain.
5	**Team structures haven’t adapted to AI**	Image 3 shows teams calling for “project assignments,” “architect senior roles,” “human mentoring,” and “focus on talent people” — indicating current team structures don’t account for deliberate learning alongside AI-accelerated delivery.

5 Concrete Team Practices to Introduce

Each practice can begin next week. They are designed to preserve AI speed while restoring the learning loops that build real engineering judgment.

Practice 1: "AI Output Review Ritual"

- **What:** Every PR that includes AI-generated code must have a "What I changed vs. what AI suggested" comment. The junior must annotate at least 3 lines explaining why they accepted or modified the AI's suggestion.
- **Owner:** Tech Lead sets up the PR template; Senior Devs enforce during code review.
- **Balance:** AI still writes the initial draft (speed preserved), but the junior must prove they read and understood it (learning restored).

Practice 2: "Weekly Debug-Without-AI Hour"

- **What:** One hour per week, the team works on a real bug or challenge with AI tools turned off. Juniors pair with seniors to walk through debugging using logs, stack traces, and documentation only.
- **Owner:** Senior Dev schedules and facilitates; Manager protects the time from meeting conflicts.
- **Balance:** 49 hours of AI-assisted work per week, 1 hour of deliberate practice. This directly addresses the "practice without AI" and "learn by yourself" themes from Image 3.

Practice 3: "Architecture Explain-Back Sessions"

- **What:** Before a junior starts a feature, they must draw or describe the relevant part of the system architecture and explain how their feature fits in. After completion, they present what actually happened vs. what they expected.
- **Owner:** Tech Lead assigns the task and reviews the explain-back; Senior Dev mentors if the junior is stuck.
- **Balance:** AI can help build the feature fast, but the junior must demonstrate system-level understanding. This addresses "system architecture" and "system design" gaps from the word clouds.

Practice 4: "Security Checkpoint in Every Sprint"

- **What:** Add a 15-minute security review item to every sprint retrospective. Pick one piece of recently shipped code (especially AI-generated), and as a team, identify at least one potential security issue — injection, auth bypass, data exposure, etc.
- **Owner:** Tech Lead rotates who presents; Manager tracks that it happens consistently.
- **Balance:** The sprint still runs at AI speed, but every two weeks the team builds the security muscle that Image 2 showed is weakening.

Practice 5: "Ownership Rotation for AI-Heavy Features"

- **What:** When a feature was largely AI-generated, assign a different junior to maintain it for the next sprint. That second developer must read, understand, and document the code — no asking AI to explain it.
- **Owner:** Manager assigns rotations; Tech Lead checks that documentation meets quality standards.

- **Balance:** The first developer shipped fast with AI (speed). The second developer builds reading and ownership skills (learning). Directly addresses “less ownership of code” and “no one knows what they do.”

3 Mistakes to Avoid When Introducing AI

Mistake 1: Banning AI Instead of Structuring It

Some teams react to AI risks by restricting access. This backfires — developers use AI anyway, just secretly, and without any review process. The workshop data shows teams want “learn how to use AI” and “build AI literacy,” not prohibition. Structure AI use; don’t forbid it.

Mistake 2: Measuring Only Velocity

If the only metric is how fast features ship, AI will always “win” — and quality, security, and learning will always lose. Add learning metrics: can the developer explain the code? Did code review catch issues? Are security vulnerabilities decreasing? The word clouds show teams worried about “surface level” work and “blind trust” — both symptoms of velocity-only measurement.

Mistake 3: Assuming Seniors Don’t Need to Adapt

AI changes senior roles too. Seniors must become better reviewers of AI output, better mentors, and better at articulating architectural reasoning — because they are now the safety net. Image 3’s emphasis on “code review,” “human mentoring,” and “architect senior roles” shows teams already feel this gap. Invest in senior development, not just junior guardrails.

30-Day Action Plan

Week	Actions	Owner	Deliverable
Week 1	Audit current AI usage: survey the team on which tasks use AI and where developers skip understanding. Set up PR template with “AI annotation” section. Schedule recurring Debug-Without-AI hour.	Tech Lead + Manager	AI usage audit report; updated PR template
Week 2	Run first Debug-Without-AI session. Introduce Architecture Explain-Back for one active feature. Identify one recent AI-generated feature for ownership rotation.	Senior Dev (debug); Tech Lead (architecture)	Session notes; first explain-back completed
Week 3	First sprint security checkpoint. Review PR annotations from Week 1–2: are juniors actually explaining AI code? Begin ownership rotation for the identified feature.	Tech Lead (security); Manager (review process)	Security finding logged; rotation assigned
Week 4	Retrospective on all five practices: what worked, what needs adjusting? Collect feedback from juniors on learning impact. Set month 2 targets based on results.	Manager (retro); all team members	Month 1 retro report; month 2 plan

SECTION B — For Junior Developers

> **A Message to You**
>
> AI will not take your job. But a developer who understands fundamentals AND uses AI effectively will outperform one who only relies on AI. Your seniors grew strong through struggle — debugging at 2am, failing in production, wrestling with system design. You need those experiences too, even when AI offers a shortcut.

5 Skills You Must Actively Seek Out

These emerged from what your senior colleagues said made them stronger developers (Image 1). AI can do these tasks for you — but doing them yourself is how you build the judgment to eventually lead teams and architect systems.

#	Skill	Why It Matters	How to Practice
1	**Debugging from scratch**	Your seniors listed “debug kernel errors,” “fixing bug at 2am,” and “fix critical bugs” as formative experiences. Debugging teaches you how systems actually work.	When you hit a bug, spend 30 minutes trying to solve it with logs and documentation before asking AI. Track what you learn.
2	**System design and architecture**	“System design,” “system analysis and design,” and “system architecture” appeared across multiple word clouds. Understanding how parts connect is what separates a coder from an engineer.	Before starting a feature, draw the system components it touches. After completing it, compare your drawing to reality.
3	**Reading and reviewing others’ code**	“Code review” was the single most prominent response in Image 3. Reading code builds pattern recognition, catches AI mistakes, and teaches coding standards.	Volunteer to review PRs even as a junior. Read the code line by line; if you don’t understand something, ask — that’s learning.
4	**Security fundamentals**	“Security” was flagged as weakened by AI usage. If you can’t recognize an SQL injection, a broken auth flow, or exposed secrets, you cannot judge whether AI output is safe to ship.	Study the OWASP Top 10. For every AI-generated endpoint, ask: “What happens if the input is malicious?” [Added beyond word cloud data]

5	**Working through failure independently**	“My first big failure,” “overcome difficulties,” “patient,” “had to do things by myself” — your seniors credit resilience as a core growth driver.	Take on one hard task per month without AI. Accept that it will be slow. The struggle is the learning.

Personal Learning Path

Months 1–6: Build Your Foundation

- **Goal:** Develop the ability to write, read, and debug code without AI assistance for core tasks.
- **Debug first, AI second:** For every bug, spend 30 minutes with stack traces, logs, and docs before using AI. Log what you tried and what worked.
- **Read code daily:** Spend 15 minutes reading a PR or open-source code. Annotate what each section does.
- **Learn one security concept per week:** Start with the OWASP Top 10. Identify each vulnerability in code you ship. [Added beyond word cloud data]
- **Build one personal project without AI:** Image 1 shows “personal website” as a formative experience. Build something from scratch to understand the full stack.

Months 6–12: Develop Judgment

- **Goal:** Move from following instructions to making engineering decisions.
- **Own a feature end-to-end:** From requirements through architecture, implementation, testing, and deployment. AI can assist, but you make every decision and can explain every choice.
- **Start reviewing AI output critically:** For every AI suggestion, write down one thing it got right and one thing that’s wrong or missing. Track your accuracy over time.
- **Study system architecture:** Learn how your application’s components connect. Draw diagrams. Ask seniors to review your understanding.
- **Present a technical topic to your team:** Teaching forces deep understanding. Image 1 shows “mentorship” as a growth moment — start by being the one who explains.

Months 12–24: Use AI as a Force Multiplier

- **Goal:** Use AI strategically, knowing when to trust it and when to override it.
- **Mentor a newer junior:** Explain architecture, review their code, help them debug. This cements your own knowledge.
- **Lead a security review:** Take ownership of reviewing AI-generated code for security issues in your team’s sprint.
- **Contribute to architectural decisions:** Propose system designs, evaluate trade-offs, and push back when AI suggestions don’t fit the system context.
- **Set AI boundaries intentionally:** Know which tasks you delegate to AI (boilerplate, syntax, documentation drafts) and which you do yourself (core logic, security-sensitive code, architectural decisions).

How to Use AI as a Learning Tool — Daily Habits

Habit	What to Do	Why It Works
The "Explain It" Rule	After AI generates code, explain every line to yourself or a colleague before committing. If you can't explain it, you don't understand it.	Transforms AI from a crutch into a teaching tool. Directly counters "blind trust in AI outputs."
The "Break It" Test	After accepting AI code, deliberately try to break it: pass null inputs, exceed limits, send unexpected types. Note what fails.	Builds edge-case awareness and testing instincts. Addresses "ignoring edge cases" and "some quick test."
The "Why Not" Question	Ask AI: "What are three alternative approaches to this, and what are the trade-offs of each?" Choose one and justify your choice.	Trains architectural thinking and decision-making. Counters "copy paste coding."
The Security Scan	Before every commit, ask yourself: "If this input came from an attacker, what would happen?" Check for injection, auth, and data exposure. [Added beyond word cloud data]	Builds the security instinct that AI cannot provide. Addresses "security" gaps from Image 2.
The Learning Log	End each day with a 2-minute note: "What did I learn today that AI didn't tell me?" Track your growing knowledge.	Creates awareness of your own growth. Supports "track learning" from Image 3.

Security and Architecture Knowledge You Need Before Trusting AI

*You cannot guide AI toward secure, scalable software if you don't know what secure and scalable looks like. The workshop identified these as critical gaps. Items marked with * are added beyond what appeared in the word clouds.*

Security Knowledge Checklist

- Understand the OWASP Top 10 vulnerabilities (injection, broken auth, XSS, etc.) *
- Know how to validate and sanitize user input in your language/framework *
- Understand authentication vs. authorization and common flaws *
- Recognize when AI-generated code exposes secrets, uses hardcoded credentials, or skips encryption *
- Know your team's security standards and how to apply them to AI output

Architecture Knowledge Checklist

- Understand how your application's components connect (databases, APIs, services)

- Know the difference between monolithic and microservice architectures and when each applies
- Understand how data flows through your system from user request to database and back
- Recognize scaling implications: what happens when 10x users hit this code? *
- Know how to evaluate whether AI-generated architecture fits your system’s constraints

SECTION C — When to Use What

Not all tasks carry the same AI risk. This guide categorizes common development tasks by how much human oversight they require. Based on workshop data plus industry security practices where marked.

☑ Green Zone — Use AI Freely

AI adds the most value with the least risk in these areas. Minimal human review needed.

Task	Why AI Is Safe Here	From Workshop Data
Boilerplate code and scaffolding	Repetitive, well-defined patterns with low risk of hidden errors	“Fast information” and speed benefits acknowledged across all images
Documentation drafts	Writing quality can be reviewed easily; errors are non-destructive	“Write a good document” listed as a developer growth area
Unit test generation	AI can generate test scaffolding quickly; tests are self-verifying	“Some quick test” noted as a shortcut risk, but test scaffolding is low risk *
Syntax lookup, language translation, formatting	Low stakes; errors are immediately visible	“Multiple languages” and “changing technologies” referenced
Prototyping and proof-of-concept	Speed is the goal; code will be rewritten for production	Speed benefits valued when code is explicitly disposable *

⚠ Yellow Zone — Use with Controls

AI helps but human review is critical. Always pair AI output with experienced review.

Task	Why Review Matters	Required Control	From Workshop Data
Feature implementation	AI may miss edge cases and business logic nuances	Senior code review with focus on edge cases and architecture fit	“Ignoring edge cases,” “surface level,” “business logic”
Bug diagnosis and fixing	AI may fix symptoms not root causes	Developer must explain the root cause before applying AI fix	“Debug kernel errors,” “fix blocker issues,” “problem-solving”
Code refactoring	AI can change structure but may not preserve behavior in all cases	Comprehensive test coverage before and after; senior review	“Code review” emphasis; “understand core” *
Database queries and data access	Performance and correctness issues can be subtle and costly	Review query plans; test with production-like data volumes *	“Lack of accuracy”; “system” *
Deployment configurations	Small errors can cause outages or security exposures	Peer review plus staging environment validation *	“Deploy in prod environment” listed as formative challenge

🛑 Red Zone — Use with Caution or Avoid

AI is dangerous without deep experience in these areas. Human judgment must lead; AI can only assist.

Task	Why AI Is Dangerous	What to Do Instead	From Workshop Data
Security-critical code (auth, encryption, access control)	AI frequently generates insecure patterns that appear to work correctly	Write manually following established security patterns; use AI only to review, not generate *	"Security" flagged as weakened by AI
System architecture decisions	AI lacks context about your specific system constraints, team capabilities, and business requirements	Senior architect leads; AI can generate options but humans must evaluate trade-offs	"System architecture," "system design" highlighted as risk areas
Business logic with legal or financial implications	Errors can have regulatory, financial, or legal consequences invisible in code review	Domain expert review required; AI cannot understand compliance context *	"Business logic" identified as AI weakness
Incident response and production debugging	AI may suggest plausible but wrong fixes under time pressure; production context is unique	Experienced engineer leads diagnosis; AI can search logs but not make judgment calls	"Deep dive client's issue," "problem-solving" require human context
Performance optimization	AI optimizes locally but may create system-wide bottlenecks or break caching strategies *	Profile first, understand bottleneck, then decide approach; AI can implement but not diagnose *	"Protect complexity exposure"; system-level thinking *

The Bottom Line

AI makes us faster. Understanding makes us reliable. Security makes us trustworthy. The teams that win will be the ones who use AI for speed while investing in the human judgment that AI cannot replace. The workshop data is clear: your teams already know what's at risk and what needs to happen. This document turns that awareness into action.

Quick Reference — Workshop Learnings at a Glance

Added after the workshop as a compact take-away. For the full analysis, see Sections A–C above.

The One-Line Summary

AI makes us faster. Understanding makes us reliable. Security makes us trustworthy. Use AI for speed — invest in the human judgment AI cannot replace.

For Seniors / Tech Leads / Managers

Action	What to Do
Structure AI, don't ban it	Require AI annotation in PRs ("what I changed vs. what AI suggested"). Forbidding AI backfires — developers use it secretly without review.
Protect learning time	Schedule 1 hour/week "Debug-Without-AI" for juniors + seniors pairing. 49 hours AI-assisted, 1 hour deliberate practice.
Enforce code review of AI output	"Code review" was the single largest word in workshop responses. AI-generated code needs more review, not less.
Add a sprint security checkpoint	15 minutes per sprint retro: pick one AI-generated piece of code and find at least one security issue as a team.
Measure learning, not just velocity	Can the developer explain the code? Did review catch issues? Are security findings decreasing? Speed alone rewards "surface level" work.
Invest in senior development too	Seniors are now the safety net for AI output. They need to become better reviewers, better mentors, and better at articulating architectural reasoning.

For Junior Developers

Skill	Start This Week
Debugging from scratch	Spend 30 min with logs and docs before asking AI. Your seniors grew strong through "fixing bugs at 2am" — you need that struggle too.
Read and review code	Volunteer to review PRs. 15 min/day reading others' code builds the pattern recognition AI can't give you.
System architecture	Before every feature, draw the system components it touches. After, compare your drawing to reality. Understanding connections is what separates a coder from an engineer.
Security fundamentals	Learn OWASP Top 10. For every AI-generated endpoint ask: "What happens if the input is malicious?" If you can't spot injection, you can't judge AI output.
Use AI as your tutor	Don't just accept AI output — ask it to explain why. Use AI as a teacher: let it walk you through concepts, challenge your understanding, and quiz you. The goal is to learn, not just to ship.

AI Usage Zones — Know When to Trust, When to Check

Zone	Tasks	Rule
Green ✓	Boilerplate, docs, unit test scaffolding, syntax, prototypes	Use AI freely — minimal review needed
Yellow ⚠	Feature code, bug fixes, refactoring, DB queries, deployment configs	Use with senior review — AI helps but humans must verify
Red 🛑	Security-critical code, architecture decisions, business logic, incident response, perf optimization	Human judgment leads — AI assists only, with deep experience required

Appendix


DEVDAY.org
Inspiring
IT Movement
AI X HUMAN
TOPIC
The Last Junior Developer: The Ticking Time Bomb in AI-Powered Teams
Sebastian Sussmann
CIO at Axon Active Vietnam
AXON ACTIVE
Focusing on your agility
AI is replacing junior developers—and we should be terrified. Every senior today was shaped by years of mistakes, late-night debugging, and hard-earned judgment. Now companies are cutting juniors and calling it transformation. But when the juniors disappear, where do future seniors come from? Sebastian exposes this dangerous paradox and, in a hands-on workshop, helps teams confront expertise debt and build an urgent upskilling blueprint—before the pipeline collapses.
Language: English
Time: 10:50 – 11:40 AM, , 18 April 2026
Room: 1156, 11th Floor
Duy Tan University, 03 Quang Trung st., Da Nang City

AhaSlides

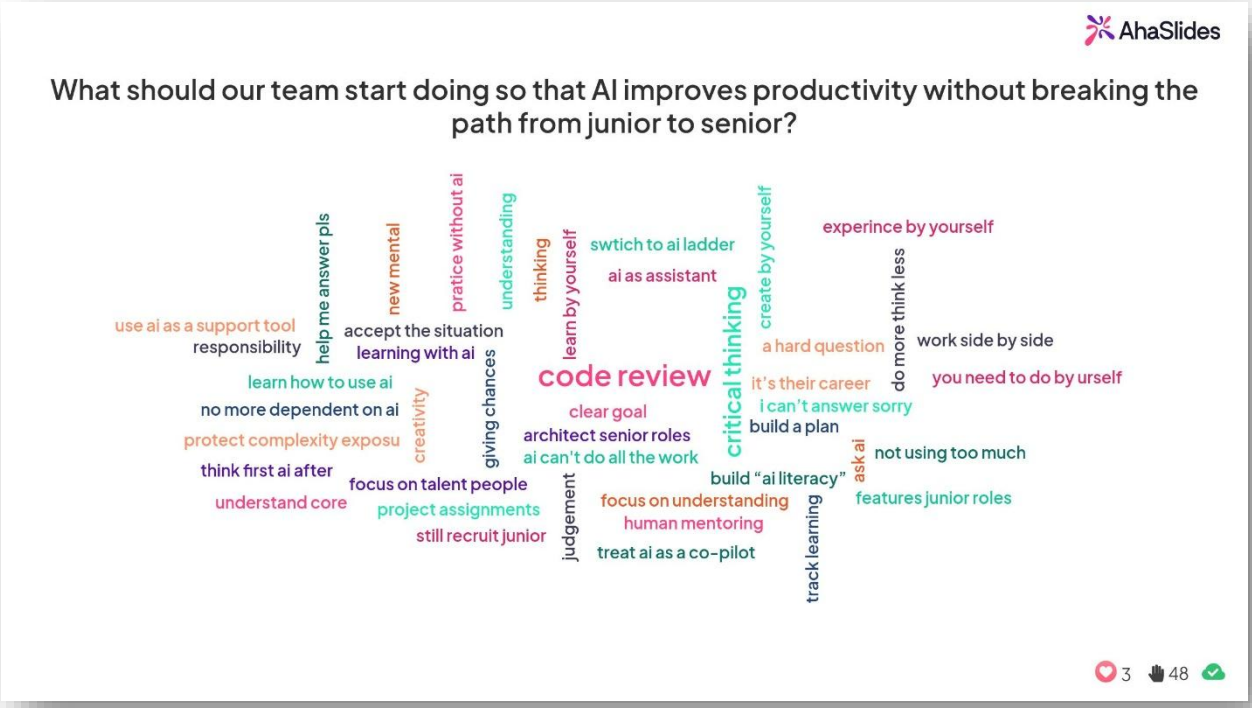
Where in our team does AI increase speed but weaken understanding, ownership, or judgment?

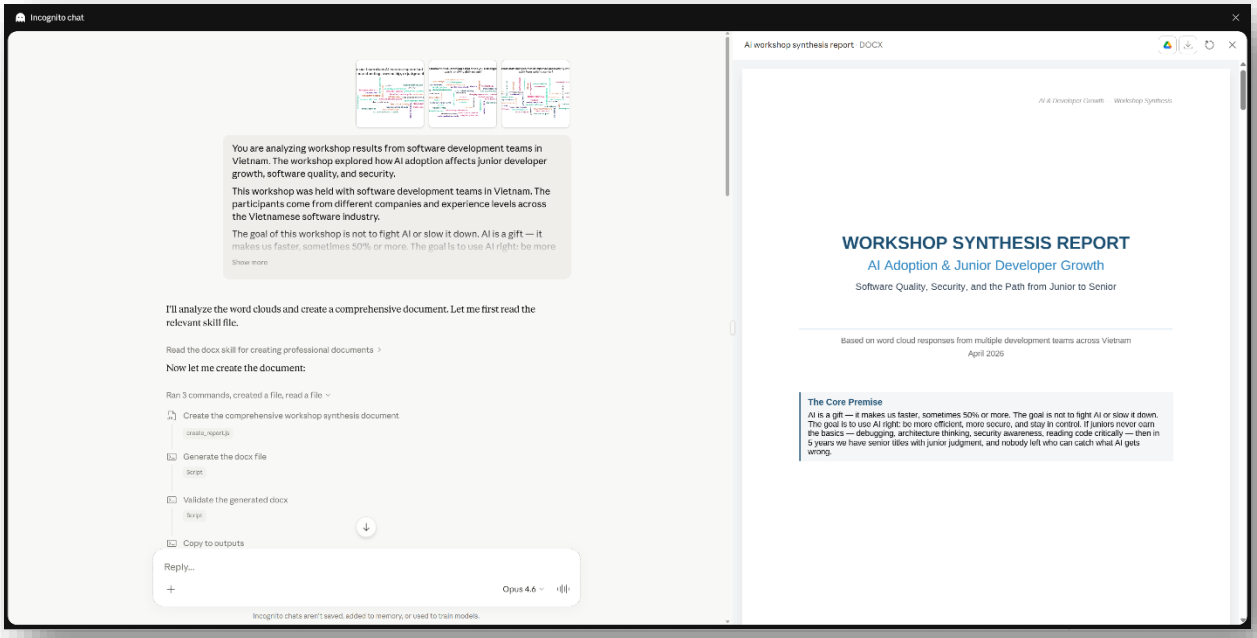
1 1 32

AhaSlides

Think of a specific moment, task, or struggle that made you a stronger developer. What was it, and why did it matter?

4 43


AhaSlides
What should our team start doing so that AI improves productivity without breaking the path from junior to senior?
use ai as a support tool
responsibility
help me answer pls
new mental
accept the situation
learning with ai
pratice without ai
understanding
thinking
learn by yourself
swtich to ai ladder
ai as assistant
create by yourself
experince by yourself
do more think less
work side by side
a hard question
you need to do by urself
it's their career
i can't answer sorry
learn how to use ai
no more dependent on ai
protect complexity exposu
think first ai after
understand core
creativity
giving chances
code review
clear goal
architect senior roles
ai can't do all the work
critical thinking
build a plan
ask ai
not using too much
build "ai literacy"
features junior roles
focus on talent people
project assignments
still recruit junior
judgement
focus on understanding
human mentoring
treat ai as a co-pilot
track learning
3
48


Incognito chat
You are analyzing workshop results from software development teams in Vietnam. The workshop explored how AI adoption affects junior developer growth, software quality, and security.
This workshop was held with software development teams in Vietnam. The participants come from different companies and experience levels across the Vietnamese software industry.
The goal of this workshop is not to fight AI or slow it down. AI is a gift — it makes us faster, sometimes 50% or more. The goal is to use AI right: be more
Show more
I'll analyze the word clouds and create a comprehensive document. Let me first read the relevant skill file.
Read the docx skill for creating professional documents
Now let me create the document:
Ran 3 commands, created a file, read a file
Create the comprehensive workshop synthesis document
Generate the docx file
Validate the generated docx
Copy to outputs
Reply...
Opus 4.6
Incognito chats aren't saved, added to memory, or used to train models.
AI workshop synthesis report · DOCX
WORKSHOP SYNTHESIS REPORT
AI Adoption & Junior Developer Growth
Software Quality, Security, and the Path from Junior to Senior
Based on word cloud responses from multiple development teams across Vietnam
April 2026
The Core Premise
AI is a gift — it makes us faster, sometimes 50% or more. The goal is not to fight AI or slow it down. The goal is to use AI right: be more efficient, more secure, and stay in control. If juniors never learn the basics — debugging, architecture thinking, security awareness, reading code critically — then in 5 years we have senior titles with junior judgment, and nobody left who can catch what AI gets wrong.