TECH RECRUITMENT 101

How to Assess Technical Depth Without Being Technical

A practical guide to hearing real engineering substance in interviews

PART OF THE *TECH RECRUITMENT 101* SERIES

How to Assess Technical Depth Without Being Technical

A practical guide to hearing real engineering substance in interviews

CONTENTS

How to Assess Technical Depth Without Being Technical

Introduction

I wrote this because I kept seeing the same hiring mistake from both sides of the table: people confusing polished technical talk with technical depth.

You do not need to become an engineer in disguise to screen engineers well. You need a better ear. After years of hiring into technical teams, I found that the most reliable signals are not jargon, speed, or confidence. They are ownership, complexity, decision-making, debugging, and trade-offs.

That matters even more now. Candidates are using artificial intelligence (AI) heavily in job search prep, and developers themselves say AI makes gaming assessments easier. At the same time, recruiters are being pushed toward more skills-based hiring, not less. According to the 2025 Future of Recruiting report from LinkedIn (https://business.linkedin.com/content/dam/me/business/en-us/talent-solutions/resources/pdfs/future-of-recruiting-2025.pdf), 93% of talent acquisition professionals say accurately assessing skills is crucial to improving quality of hire. In the 2025 Recruiter Nation Report from Employ (https://pages.employinc.com/rs/659-JST-226/images/2025-Employ-Recruiter-Nation-Report.pdf), half of HR leaders say it is difficult to measure or validate skills consistently. That is the gap this book is meant to help with.

This is one book in a broader series I have written for recruiters who want to get sharper without turning every screen into theater. Here I am focused on one practical promise: I will show you how to recognize genuine technical depth by listening for ownership, complexity, decision-making, debugging, and trade-offs, so you can run better screens, ask sharper follow-ups, and stop confusing confidence with substance.

Why technical depth is hard to hear at first

The first reset is simple: your job is not to verify code. Your job is to hear whether the person in front of you has actually carried technical responsibility.

Many recruiters are trained to reward surface fluency: polished explanations, trendy keywords, quick answers, and a tidy career story. Unfortunately, those are weak signals. Deep engineers often pause, qualify, correct themselves, or answer with "it depended." That is not thrilling conversation, but it is often a sign they have worked through real constraints. Smooth candidates, meanwhile, can talk for half an hour and leave you with nothing but a nice glossary.

That mismatch matters more now because polished performance is easier to manufacture. The 2025 Job Seeker Nation Report from Employ (https://pages.lever.co/rs/659-JST-226/images/2025-Job-Seeker-Nation-Report.pdf) found that candidates already use AI in job search and interview prep, while the 2025 Developer Skills Report from HackerRank (https://www.hackerrank.com/reports/developer-skills-report-2025) found that 76% of developers think AI makes gaming assessments easier. The same HackerRank report found that 96% say problem-solving should matter more than memorization. I agree. Recall is loud. Judgment is quieter.

So what is technical depth in hiring terms? Not brilliance in the abstract, and not vocabulary. It is evidence that someone has dealt with complexity they could not simply talk around. In practice, it shows up in five places: what they owned, what made the work difficult, what choices they made, how they investigated problems, and what trade-offs they accepted.

When I screen for depth, I listen less for the tools and more for the decisions around them. "We used Kubernetes" tells me very little. Kubernetes is software for running applications across many servers. The useful signal is what comes next: why they used it, what problems it created, what failed, what they changed, and how they knew it worked. Strong engineers usually have opinions shaped by consequences.

A common mistake is treating confidence as proof of competence. Confidence can help, but it can also be theater. Some excellent engineers are understated, awkward, or simply careful. I have seen very capable people describe a serious technical rescue as "fixing some performance issues." Your follow-up questions have to do the lifting.

> 💡 **A better listening target**
>
> Do not ask yourself, "Did this sound smart?" Ask, "Did I hear evidence of real responsibility, concrete decisions, and proof of outcomes?"

One more modern wrinkle is AI itself. Saying "I use AI all the time" is not a strong signal. The better signal is whether the candidate sounds appropriately careful about verification. In the 2025 Stack Overflow Developer Survey AI section (https://survey.stackoverflow.co/2025/ai), more developers distrusted AI output accuracy than trusted it, and only a small minority reported highly trusting the output. Strong engineers tend to sound specific about where AI helped, where it failed, and how they checked the result before shipping anything that could wake people up at 3 a.m.

The five signals I listen for

If this book has a backbone, this is it.

When I screen engineers, I listen for five recurring signals: ownership, system complexity, decision-making, debugging, and trade-offs. Different roles weight them differently, but the structure holds across backend, frontend, infrastructure, data, security, and AI-flavored work.

The framework matters even more in an AI-shaped market. According to the 2025 Developer Skills Report from HackerRank (https://www.hackerrank.com/reports/developer-skills-report-2025), 78% of developers say hiring assessments do not align with real-world tasks, 66% prefer practical coding challenges, and 96% say problem-solving should matter more than memorization. Surface performance is getting cheaper. Substance is not.

Technical depth signal checklist

✓ Ownership: What did they personally drive?

✓ System complexity: Can they explain what made the problem genuinely hard?

✓ Decision-making: Can they walk through why one path was chosen?

✓ Debugging: Can they describe how they investigated uncertainty?

✓ Trade-offs: Do they understand the costs and constraints behind the choice?

Ownership

Ownership clears away a lot of fog. Strong candidates can separate their contribution from the team's work without sounding possessive about it. They can tell you what they were responsible for, what decision was theirs, and where their responsibility stopped.

Weak answers stay blurred: "we built," "we migrated," "we improved performance." Team language by itself is not a problem. The question is whether they can zoom in when asked. "What

part was yours?" is one of the highest-return questions in technical recruiting.

System complexity

I am not looking for a list of tools. I am listening for the shape of a real problem.

Complexity sounds like scale, latency, messy data, synchronization, reliability, security, model quality, cost, or conflicting business constraints. It sounds like interacting pressures, not a fashionable architecture diagram. New tools can make a story sound sophisticated. That does not mean the candidate dealt with a hard system.

Decision-making

Technical depth shows up in choices. Good engineers can explain what options they considered, what mattered, and why the final decision fit the situation.

I trust candidates who can explain an imperfect decision calmly. The ones who present every past choice as obviously correct are usually giving you a cleaned-up story.

Debugging

Debugging is where a great deal of real engineering hides.

I do not mean only fixing bugs. I mean how someone investigates uncertainty when the cause is not obvious. Strong candidates can explain what signals they checked, what hypotheses they tested, and how they knew they were done.

This matters even more now because AI-generated code often creates problems that look plausible before they break. In the 2025 Stack Overflow Developer Survey AI section (https://survey.stackoverflow.co/2025/ai), 66% of developers said the biggest frustration with AI tools is getting solutions that are almost right, and 45% said debugging AI-generated code takes more time. That is why debugging has become such a strong depth signal.

Trade-offs

Mature engineers know there was no perfect answer. There was a choice made under constraints.

Time, performance, maintainability, reliability, team skill, user impact, and cost all pull in different directions. Strong candidates understand what they optimized for and what they knowingly gave up. If someone can name the compromise plainly, I trust their depth more.

How to ask follow-ups that reveal real depth

A broad answer is not the problem. Stopping there is the problem.

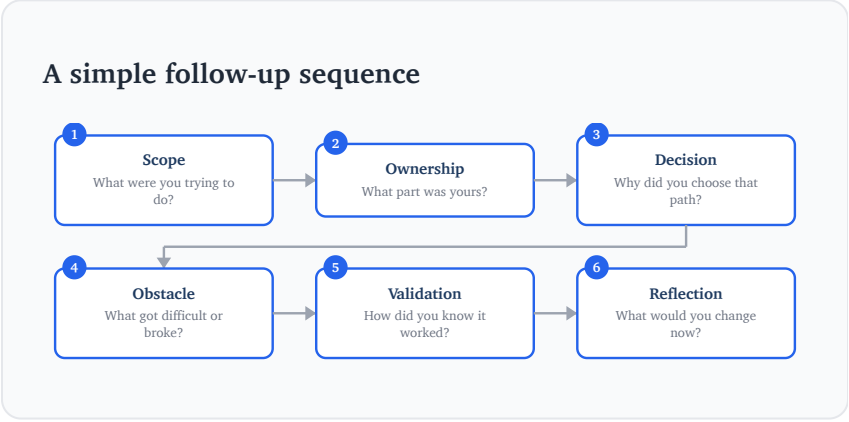
Most candidates begin with the polished version. That is normal. Many are rehearsed, and some are AI-polished too. The 2025 Job Seeker Nation Report from Employ (https://pages.lever.co/rs/659-JST-226/images/2025-Job-Seeker-Nation-Report.pdf) found that candidates already use AI for interview preparation, and the 2025 Developer Skills Report from HackerRank (https://www.hackerrank.com/reports/developer-skills-report-2025) found that developers think AI makes assessment gaming easier. Your job is not to reward the smooth first answer. Your job is to move from summary to evidence.

The good news is that you do not need to sound more technical to do this well. In fact, pretending usually makes the interview worse. The strongest follow-ups are simple:

- What part of that was yours?
- What made it hard?
- How did you decide?
- What broke first?
- How did you know the fix worked?
- What would you do differently now?

These questions do not require you to judge whether a design was elegant. They ask for evidence of lived work. A person who did the work can usually tell you where the pain was, what they owned, what they chose, and how they verified the result.

If the candidate stays high level, narrow the frame. Ask for one feature, one incident, one migration, one release, or one bug. Vague candidates prefer the aerial view. Depth lives closer to the ground.



I would also avoid over-indexing on definitions from memory. In the 2025 Developer Skills Report from HackerRank (https://www.hackerrank.com/reports/developer-skills-report-2025), 78% of developers said hiring assessments do not align with real work. That tracks with what I have seen. Engineers rarely create value by reciting textbook definitions on demand. They create value by making decisions under constraints and fixing messy things when reality is rude.

A good follow-up can turn a generic story into a useful one very quickly. I once asked a candidate who had given me a very polished answer about performance work, "What broke first?" That was the moment the real signal appeared. They stopped presenting and started explaining.

How weak and strong follow-ups sound in practice

PROMPT	WEAK ANSWER	STRONG ANSWER
What part was yours?	I worked on the platform team.	I owned the migration plan and rollback process.
What made it hard?	It was a complex system.	Two services had conflicting data assumptions.
How did you decide?	We chose the best option.	We picked the slower path because it reduced failure risk.
How did you know it worked?	It performed better.	Error rate dropped and repeat tickets stopped.

If AI tools come up, the useful question is rarely "Which tool?" It is "What did you still need to verify yourself?" Strong engineers tend to sound specific about validation, failure cases, and where

they did not trust the output blindly.

> ☝️ **A simple rule for live screens**
>
> When you hear a broad claim, ask for one concrete example. When you hear a concrete example, ask what the candidate decided, what got hard, and how they verified the result.

What strong answers sound like across modern technical roles

You do not need a different screening brain for every engineering title. The same framework still works. You are listening for ownership, complexity, decisions, debugging, and trade-offs. What changes by role is the form those signals take.

That matters because titles are noisy, and AI has made them noisier. According to LinkedIn's AI Labor Market Update from September 2025 (https://economicgraph.linkedin.com/content/dam/me/economicgraph/en-us/PDF/ai-labor-market-update-header-sept-2025.pdf), AI engineering hiring grew by more than 25% year over year in 2025. Dice also reported (https://www.dice.com/career-advice/50-of-tech-jobs-now-require-ai-skills-what-this-means-for-your-job-search-in-2026) that by September 2025, half of U.S. tech job postings required AI skills. So yes, you will hear more AI language in ordinary screens. No, that does not make the old signals obsolete.

What helps is learning what meaningful complexity sounds like in each area.

What depth tends to sound like by role

ROLE	STRONG ANSWER USUALLY INCLUDES	WEAK ANSWER USUALLY SOUNDS LIKE
Backend engineer	Reliability, scaling, data models, consistency, failure handling	Framework list, vague application programming interface (API) talk, no production impact
Frontend engineer	Performance, state, browser constraints, accessibility, user trade-offs	Library names, pretty interface talk, little user impact
Infrastructure or DevOps engineer	Automation, monitoring, incidents, recovery, capacity, risk	Tool shopping, setup steps, no failure discussion
Data engineer	Pipeline reliability, data quality, lineage, latency, ownership	Extract, Transform, Load (ETL) buzzwords, dashboards, unclear sources
Security engineer	Threats, controls, response, prioritization, realistic trade-offs	Compliance-only language, generic best practices
AI or machine learning	Problem framing, data quality,	Large language model (LLM) talk, fine-tuning

ROLE	STRONG ANSWER USUALLY INCLUDES	WEAK ANSWER USUALLY SOUNDS LIKE
engineer	evaluation, deployment behavior, limits	claims, no metrics or outcomes

For backend roles, listen for systems under load: bottlenecks, retries, timeouts, schema choices, data integrity, and production consequences. For frontend roles, listen for constraints in the browser, page performance, state complexity, accessibility, and user experience trade-offs. Frontend engineers often sound more like systems thinkers than recruiters expect. The browser has humbled many confident adults.

For infrastructure roles, including Site Reliability Engineers (SREs), the people responsible for keeping production systems reliable, listen for operational judgment: what failed, how it was detected, how repeat incidents were reduced, and what was automated versus left manual for safety.

For data roles, strong candidates talk about data quality, pipeline failures, freshness, ownership across teams, and what decisions depended on the data. For security roles, listen for threat thinking and prioritization, not just policy language.

AI and machine learning roles need the same framework applied carefully. The 2025 Developer Skills Report from HackerRank (https://www.hackerrank.com/reports/developer-skills-report-2025) reports that developers now use AI broadly, and the 2025 Stack Overflow Developer Survey AI section (https://survey.stackoverflow.co/2025/ai) shows that many still distrust the output. So the useful question is not whether AI appeared in the project. It is whether the candidate understood and owned the work.

For AI-related roles, strong answers usually include six things: the problem they were solving, the data they used, how they evaluated quality, what production constraints existed, how the system behaved after release, and what they changed when reality disagreed with the demo. If someone says they built an LLM feature, that tells you almost nothing. If they can explain how they handled low-confidence outputs, evaluation, latency, monitoring, and fallback to humans, you are hearing something real.

> ☝️ **A simple rule for any technical role**
>
> Do not reward trendy words. Reward clear ownership, specific constraints, measurable outcomes, and thoughtful trade-offs.

Red flags, false positives, and when to trust your doubt

A lot of screening mistakes come from treating confidence as proof and uncertainty as failure. In practice, both are noisy.

The most common false positive is the polished storyteller who stays generic. Everything sounds impressive until you ask what they personally changed, why they chose that approach, what broke, or what trade-off they accepted. Another is borrowed ownership: plenty of "we built" with very little evidence of what was actually theirs. A third is tool fluency without decision logic, especially around AI. The 2025 Developer Skills Report from HackerRank (https://www.hackerrank.com/reports/developer-skills-report-2025) found that 76% of developers say AI makes gaming assessments easier. That should make every recruiter a little less dazzled by smoothness.

False negatives matter just as much. Quiet candidates, non-native English speakers, and engineers from less famous companies are often underestimated because they do not package their work dramatically. I have hired excellent people who were unimpressive only in the very narrow sense that they did not sound like conference speakers. That turned out to be completely survivable.

One habit helps with both errors: write down exactly what felt thin. Not "something was off." Name the gap in plain language. Did not explain why the design was chosen. Stayed vague on debugging. Used team language but never named personal contribution. Named tools without describing constraints. Once you write the gap clearly, the next follow-up usually becomes obvious.

This is also where recruiters can get tangled up with their own uncertainty. "I do not understand this answer" is not the same as "this answer lacks substance." If the answer is specific but technical, ask for plain English. If the answer is fluent but vague, ask for detail.

When doubt means something, and when it does not

WHAT YOU HEARD	LIKELY ISSUE	BEST FOLLOW-UP
Complex answer you do not follow	Your knowledge gap	Ask for plain-English impact
Smooth answer with no specifics	Weak substance	Ask for one concrete example
Repeated 'we' with no clear role	Unclear ownership	Ask what they personally owned
Tool names without decision logic	Keyword fluency	Ask why that tool was chosen

A few grounded follow-ups go a long way:

- What was the hardest part of that?
- What did you decide yourself?
- What alternatives did you consider?
- What went wrong the first time?
- How did you verify the fix?
- If I asked your manager what part you owned, what would they say?

Usually one good follow-up is enough. You are not trying to corner the candidate. You are trying to find out whether your doubt points to a real gap or just an incomplete first answer.

When the signal is mixed but important, escalate to the hiring manager. That is not weak screening. That is good judgment about the limits of your screen.

And when you pass signal into a debrief, keep it clean. Separate observations from conclusions. "Could not explain why they chose this database" is useful. "Not technical enough" is often lazy. "Spoke generally about AI-assisted coding but clearly explained how they tested generated code before shipping" is useful. "Seemed good with AI" is not. In the 2025 Stack Overflow Developer Survey AI section (https://survey.stackoverflow.co/2025/ai), 84% of developers said they use or plan to use AI tools, but only a small minority highly trust the output. Discernment is a better signal than enthusiasm.

> ⚠ **Do not convert uncertainty into certainty**
>
> If you are unsure, say you are unsure. Name the exact gap, ask one more grounded question, or escalate. Invented certainty sounds decisive in the moment and damages hiring quality later.

Conclusion

If you take one thing from this book, let it be this: you do not need to sound technical to assess technical depth well. You need a consistent way to listen.

In my experience, the most reliable signals are still the least theatrical ones. Ownership. Complexity. Decision-making. Debugging. Trade-offs. If you can hear those clearly, you will run better screens, write better debriefs, and waste less time being impressed by the wrong things.

That matters in a market where skills-based hiring is becoming the norm, AI is making polished answers cheaper, and recruiters are still the people candidates trust most in the live process. The 2025 Job Seeker Nation Report from Employ (https://pages.lever.co/rs/659-JST-226/images/2025-Job-Seeker-Nation-Report.pdf) found that candidates trust HR staff more than AI to guide them through interviews. Good. We should earn that.

This book is part of a larger set I have written for recruiters who want a sharper, more grounded way to work with technical talent. If this one helps you ask better follow-ups and trust better signals, it has done its job.

Glossary

- **AI (Artificial Intelligence)** — Software that can generate, classify, predict, or assist with work that normally requires human judgment.
- **API (Application Programming Interface)** — A defined way for one software system to communicate with another.
- **Backend engineer** — An engineer who works on server-side systems, data handling, and business logic behind an application.
- **Browser** — The software people use to access websites and web applications, such as Chrome or Safari.
- **Data lineage** — A record of where data came from, how it changed, and where it moved.
- **Database schema** — The structure of how data is organized in a database.
- **Debugging** — The process of investigating problems, finding causes, and fixing them.
- **Deployment** — Releasing software into a live environment where users or internal teams can use it.
- **DevOps engineer** — An engineer focused on how software is built, released, and operated reliably.
- **ETL (Extract, Transform, Load)** — A common data workflow for moving data from one place to another, cleaning or reshaping it along the way.

- **Fallback** — A backup path the system uses when the primary approach fails.
- **Fine-tuning** — Additional training of an AI model on specific data to make it better at a narrow task.
- **Frontend engineer** — An engineer who builds the part of an application that users see and interact with.
- **Hallucination** — In AI, an output that sounds plausible but is false or unsupported.
- **Infrastructure engineer** — An engineer who works on the underlying systems, environments, and tooling that keep software running.
- **Kubernetes** — A system for running and managing applications across many servers.
- **Latency** — Delay or response time in a system.
- **LLM (Large Language Model)** — A type of AI model trained on large amounts of text and used for tasks like answering questions or generating content.
- **Machine learning** — A branch of AI where systems learn patterns from data to make predictions or decisions.
- **Model evaluation** — The process of checking how well an AI or machine learning system performs.
- **Monitoring** — Watching system behavior through metrics, logs, or alerts to detect problems.
- **Pipeline** — A sequence of steps that moves data or software changes from one stage to another.

- **Production** — The live environment where real users or business operations depend on the system.
- **Rollback** — Reverting a system change to a previous version after a problem.
- **Schema** — The structure or format used to organize data.
- **Site Reliability Engineer (SRE)** — An engineer focused on keeping production systems reliable and available.
- **State management** — How an application keeps track of current data, user actions, and changes over time.
- **Timeout** — A failure that happens when a system takes too long to respond.
- **Tool fluency** — The ability to talk comfortably about tools, sometimes without showing real depth of use.
- **Trade-off** — A choice where improving one thing requires accepting a cost somewhere else.
- **User experience** — How a product feels to use, including clarity, speed, accessibility, and ease of use.
- **Validation** — Checking whether a result, fix, or output is actually correct and reliable.

About This Book

About the Author

Natalia Romanova

Natalia is the founder of Talantrix, an applicant tracking system for tech recruiters. She began her career as a software engineer, later moved into engineering management, and eventually led several software companies, including ones she founded. Along the way she repeatedly had to build technical recruiting capability, and over time ended up learning a lot about hiring engineers—more than she ever expected to.

About Talantrix

Talantrix is an intelligent applicant tracking system built by recruiters, for recruiters. Designed specifically for technical hiring, it helps recruitment teams automate the busywork — from CV parsing and skill-based search to pipeline management — so they can focus on what matters most: connecting with top engineers and making great hires.

Learn more at https://talantrix.com (https://talantrix.com)

The Tech Recruitment 101 Series

A practical book series for recruiters who work with engineering teams. Each volume tackles a specific challenge in technical hiring — from screening candidates to understanding code — with clear explanations, real-world examples, and actionable frameworks.

Revision 1.0 — 18 March 2026

Tech Recruitment 101

A practical book series for recruiters who work with engineering teams. Each volume tackles a specific challenge in technical hiring — from screening candidates to understanding code — with clear explanations, real-world examples, and actionable frameworks.