TECH RECRUITMENT 101

How to Interview Developers

What the recruiter conversation should actually establish before engineering interviews begin

PART OF THE *TECH RECRUITMENT 101* SERIES

How to Interview Developers

What the recruiter conversation should actually establish before engineering interviews begin

CONTENTS

How to Interview Developers

Introduction

I wrote this book because I kept seeing the same problem from both sides of the hiring table: recruiter screens treated as admin, then engineering interviews used to discover things that should have been clear much earlier.

From my years in engineering, management, and running software companies, recruiting was never separate from the work. I had to figure out who was genuinely relevant, who actually wanted the role, and where a process was going to fall apart if nobody asked the obvious questions early.

That is what this book is for. Not to turn recruiters into engineers, and not to give you a bag of trivia questions disguised as "technical screening." The recruiter interview has a narrower job, but a very important one: decide whether it makes sense to spend engineering time here.

The framework is simple. Before technical rounds start, you need enough signal on four things: real fit for the role, motivation to move, communication and collaboration, and practical process alignment. If you can evaluate those consistently, you will run better screens, write better debriefs, and stop sending people forward on optimism alone.

This book is part of a broader series on hiring developers. The others go deeper into adjacent parts of the process. This one is about the recruiter conversation itself: what to ask, what to listen for, and when not to move a candidate forward.

What the recruiter interview is actually for

A recruiter screen is not a calendar exercise. It is a decision step.

In my experience, weak hiring processes often start with the same bad assumption: the recruiter call is there to confirm the candidate is alive, interested, and available for the “real” interview later. That is how engineering teams lose time on people who were never a sensible match, and how candidates end up in processes that should have stopped much earlier.

Your job is also not to run a lightweight technical interview. You do not need to prove that you can out-engineer the engineer. You are there to establish four things early, cheaply, and clearly, before scarce technical interview time is involved.

First, is the candidate directionally relevant to the role?

Second, do their reasons for moving make sense, and does this role plausibly fit them?

Third, can they communicate and collaborate in the way this team needs?

Fourth, do the practical conditions line up well enough to proceed?

That distinction matters to candidates too. In Employ / Lever's 2025 Job Seeker Nation Report (https://pages.lever.co/rs/659-JST-226/images/2025-Job-Seeker-Nation-Report.pdf), 43% of respondents said the application process had the biggest impact on their impression of a company, 35% said they would abandon an application if it took too long, and 51% said recruiter communication, easy applications, and flexible scheduling were major contributors to a positive experience.

> ✓ **The four areas to evaluate**
>
> Before engineering interviews, establish: directional relevance, motivation to move, communication and collaboration fit, and practical process alignment.

Notice what is not on the list. You are not responsible for proving deep technical competence. You are not making the final hiring decision. You are deciding whether this person is worth the engineering team's time, and whether the candidate has enough clarity to want that next step too.

A structured recruiter screen protects both sides. It keeps engineers focused on candidates with a real chance of success, and it makes the process feel intentional to the candidate. That same Employ / Lever report (https://pages.lever.co/rs/659-JST-226/images/2025-Job-Seeker-Nation-Report.pdf) found that interviewer focus,

strong questions, conversation quality, responsiveness, and transparency all shape employer perception. The recruiter screen is part of the product whether we admit it or not.

Directional fit: is this person relevant to the actual job?

The first job of the recruiter screen is not to prove whether someone can code. It is to decide whether their experience is relevant enough to justify a deeper assessment.

What works best for me is simple: listen for scope, context, ownership, and recency. You are trying to establish whether this person has done work that maps closely enough to the role in front of you that a technical interview is worth having.

Start with the role, not the résumé. Before the call, translate the job description into a few usable screening signals. Usually that means: what kind of systems they worked on, how much ownership they had, how close they were to production, whether the environment is similar, and whether the experience is recent enough to matter.

If the role is for a backend engineer, meaning someone who builds the server-side systems behind an application, working on distributed systems, meaning software that runs across multiple machines and has to handle coordination and failure cleanly, you do not need every framework they have touched. You need to know whether they built or operated backend services, whether those systems had meaningful complexity, whether they owned decisions or mostly implemented tasks, and whether this happened recently.

These are the questions I use most often:

Directional-fit questions

- ✓ What have you built most recently?
- ✓ What part did you personally own?
- ✓ Was it in production, and who used it?
- ✓ What kind of system or environment was it?
- ✓ How does that compare to the role you want now?

Notice what is missing: trivia. “Tell me about what you built” is far more useful than “Do you know Kubernetes?” Kubernetes is a platform for deploying and managing software applications across servers. If they say yes, you still know almost nothing. If they explain they managed deployment pipelines and cluster configuration for production services, now you have context.

The same applies to ownership. “I worked on a machine learning platform” can mean they designed data pipelines, trained and deployed models, and monitored them in production. It can also mean they integrated one endpoint and sat nearby while the actual machine learning engineers did the hard part. You need to separate team proximity from personal contribution.

This matters even more now that AI-assisted applications are normal. Gartner reported in 2025 (https://www.gartner.com/en/newsroom/press-releases/2025-07-31-gartner-survey-shows-just-26-percent-of-job-applicants-trust-ai-will-fairly-evaluate-them) that many candidates use AI during the application process, including to generate résumé text and other materials. That does not make polished materials useless. It does make them cheaper. You need grounded specifics: what they built, what they owned, what constraints mattered, and what changed because of their work.

AI-heavy titles need the same treatment. “AI engineer” can mean anything from building retrieval pipelines, meaning systems that fetch relevant information for an AI application, to wrapping a large language model application programming interface, or API, which is a way one software service talks to another. Both can be legitimate experience. They are not interchangeable hires.

Because AI tools are now normal in developer workflows, I also ask how candidates used them in practice. GitHub’s 2025 developer research (https://github.blog/news-insights/research/survey-ai-wave-grows/) found usage is nearly universal, and Stack Overflow’s 2025 Developer Survey (https://stackoverflow.blog/2025/12/29/developers-remain-willing-but-reluctant-to-use-ai-the-2025-developer-survey-results-are-here/) shows developers often use AI heavily while trusting it selectively. So “uses AI” is not a signal. Judgment is.

Adjacent versus qualified

SIGNAL	ADJACENT	QUALIFIED
System exposure	Worked near it	Built or owned part of it
Production depth	Prototype or internal demo	Live system with users
Ownership	Task execution	Decision-making and delivery
Recency	Older experience	Recent repeated experience

Get comfortable saying “adjacent” instead of forcing a yes or no. Adjacent is not bad. It just means not strong enough for this role unless the hiring manager wants to stretch.

Good notes matter here. Do not write a transcript. Write a map: what they built, level of ownership, production context, relevant technologies, gaps, and what engineering should test next. If your notes just paraphrase the résumé, you are creating work, not reducing it.

Motivation: why are they moving, and why this role?

Motivation is where a lot of processes quietly break. You can have a candidate with relevant experience and workable logistics, then lose weeks because nobody established whether they actually want this job.

I do not mean whether they are “open.” I mean whether there is a real reason to move, and whether your role answers it.

“Just exploring” is not an automatic no. Good developers are often passive until something specific catches their attention. What matters is whether they can describe what would make a move real.

Start with a simple question: why now? Then keep going until the answer has shape. “I want growth” is too vague to score. Growth into what? Bigger systems, more ownership, a healthier team, less operational drag, more product influence, a different domain?

What I listen for is a concrete source of tension and a concrete pull toward something better. Push factors are what they want to leave. Pull factors are what they want to move toward. A candidate who has both is usually much easier to assess and much easier to close.

These follow-ups tend to work well:

- What is missing in your current role?
- If you moved this year, what would need to improve?
- What would make you say no, even if the compensation was good?
- How are you prioritizing compensation, scope, and flexibility?

Those questions tell you whether the candidate has standards, whether they know what they want, and whether your role can honestly meet it.

What to listen for in motivation

SIGNAL	WEAK ANSWER	USEFUL ANSWER
Reason for moving	Just exploring	Current role is too operational
Desired change	More growth	Wants system design ownership
Role interest	Sounds interesting	Matches domain and team stage
Decision criteria	Depends	Needs stronger scope and fewer meetings

Once you understand the motive, connect it to the role directly. Do not sell in the abstract. Test the match. If they want mentorship and your team expects immediate independence, say that. If they want deep backend work and the role is drifting toward coordination, say that too.

That honesty matters. Employ's 2025 benchmark reporting, summarized by HR Dive (https://www.hrdive.com/news/hiring-benchmarks-report-employ-2025-more-applicants/809604/), noted that early attrition often comes from mismatch in the hiring process or from interview promises not matching the role reality. Overselling the job does not help you close. It just delays the disappointment.

AI adds one more wrinkle. Some candidates are chasing anything with "AI" in the title because it sounds current. Others are trying to avoid shallow branding and want real product, infrastructure, or applied machine learning work. Ask what kind of AI work they mean. A large language model prototype is not the same thing as production machine learning infrastructure, and neither is the same as a chatbot someone added to a roadmap because leadership got excited on a flight.

> ☝ **A useful test**
>
> By the end of this part of the screen, you should be able to finish this sentence: this candidate would move for us if we can offer ___. If you cannot, you do not yet understand their motivation.

One final note: do not confuse urgency with seriousness. Some of the best candidates are thoughtful, not frantic. If you create clarity, you often create commitment.

Communication and collaboration: how do they work with other people?

A recruiter screen should tell you more than whether someone has touched Java, Python, or React. It should tell you how they make their work understandable to other people.

I pay close attention to whether a candidate can explain one recent project clearly: the goal, the constraints, their role, the trade-offs, and the outcome. I do not need a perfect presentation. I need a legible one.

This is not soft fluff. Teams break on communication failures, ownership confusion, and inability to work through disagreement at least as often as they break on missing syntax knowledge.

The simplest test is explanation. Ask the candidate to describe a recent project to a smart non-engineer. If the first answer is thin, help them. Some good developers speak in compressed form because they are used to talking with other engineers. Better prompts often produce much better signal.

What you want is grounded thinking. Strong answers usually include constraints, dependencies, and trade-offs. Weak answers sound polished but frictionless. Everything was important, strategic, and successful, yet you still cannot tell what they actually did.

Ownership is especially useful here. I do not want hero stories. I want candidates who can separate their own contribution from team effort. “I led the migration plan, but another engineer owned the database changes” is a healthy answer. “I basically drove everything” is often less impressive than people think.

Listen to how they talk about teammates too. Do they give credit naturally? Do they describe product managers, designers, quality assurance staff, meaning the people responsible for testing and release confidence, and engineering leaders like colleagues rather than obstacles? Contempt leaks. So does maturity.

Conflict and mistakes are both worth probing. Ask about a disagreement with a teammate or a decision they would revisit. I am listening for whether they can explain the substance of the issue, how it was handled, and what they learned. One short, specific answer usually tells you more than five minutes of polished self-protection.

AI belongs in this section too because it changes how developers work. GitHub’s 2025 research (https://github.blog/news-insights/research/survey-ai-wave-grows/) found that AI coding tools are now standard in many workflows, while Stack Overflow’s 2025 Developer Survey (https://stackoverflow.blog/2025/12/29/developers-remain-willing-but-reluctant-to-use-ai-the-2025-developer-survey-results-are-here/) shows developers often spend time correcting “almost right” output. So do not ask whether they use AI. Ask what they use it for, what they would not trust it with, and how they verify results.

That also helps with suspiciously polished answers. I am not assuming fraud every time someone sounds prepared. But if a story stays vague when you ask about trade-offs, failure points, or what happened after launch, treat the smoothness as cosmetic until proven otherwise.

Signals worth collecting here

- ✓ Can explain recent work in plain language
- ✓ Separates personal ownership from team effort
- ✓ Describes trade-offs, not just outcomes
- ✓ Talks about teammates with respect
- ✓ Handles conflict and feedback without drama
- ✓ Can discuss mistakes without a defensive speech
- ✓ Shows judgment in how AI tools are used

A final caution: polished is not the same as strong, and awkward is not the same as weak. Your job is not to reward style. It is to gather evidence about how this person functions on a real team.

Practical alignment: should this process move forward at all?

This is the least glamorous part of the recruiter conversation and one of the most important. A candidate can be strong and credible, and still be the wrong person to move forward if the practical conditions do not line up.

I ask these questions earlier than many recruiters do, but not abruptly. The tone matters. You are not interrogating someone about logistics. You are checking whether the process is worth both sides investing in.

Start with compensation and be direct. Ask what range would make a move sensible, what that number is based on, and whether they mean base salary or total compensation. A lot of mismatch is not really about money. It is about people using different definitions while pretending they are aligned.

Then get clear on location and work pattern: remote, hybrid, office expectations, relocation, travel, and time-zone overlap. “Flexible” is one of those words that causes a surprising amount of damage. At one company I worked with, it meant “some office time.” To candidates, it often meant “I can live where I like.” Those are not the same sentence.

Notice period, start timing, and decision timing matter too. Ask when they could realistically start, not just what their contract says. Ask what timeline they need from you. If your process takes weeks and they need to decide much sooner, that is not a detail to tidy up later.

That is especially important in software hiring. Employ's 2025 hiring benchmarks, summarized by HR Dive (https://www.hrdive.com/news/hiring-benchmarks-report-employ-2025-more-applicants/809604/), found that software and tech had the lowest tracked offer acceptance rate and that hiring timelines were long enough to create real risk on their own. Weak calibration early wastes scarce engineering time and increases the odds of losing the candidate later.

You should also ask about competing processes without sounding territorial. Where else are they in process? Are any stages time-sensitive? What would make them prioritize one opportunity over another? You are trying to understand urgency and decision criteria, not collect gossip.

Authorization and compliance constraints belong here too. If the role has work authorization requirements, client restrictions, export controls, or background check constraints, ask now. The same goes for role-specific realities such as on-call work, meaning being available outside normal hours to respond to production issues, or travel expectations.

This is also the point where you set expectations for technical rounds. Tell the candidate what the process is meant to evaluate, roughly how long it will take, what format to expect, and what preparation is useful. Candidates increasingly distrust black-box hiring, especially where AI is involved. Gartner's 2025 survey (https://www.gartner.com/en/newsroom/press-releases/2025-07-31-gartner-survey-shows-just-26-percent-of-job-applicants-trust-ai-will-fairly-evaluate-them) found only 26% of applicants trusted AI to evaluate them fairly. A clear human explanation goes a long way.

Practical alignment questions to settle before engineering rounds

- ✓ Compensation range and how the candidate defines it
- ✓ Location, remote policy, travel, and schedule constraints
- ✓ Notice period, realistic start date, and decision timeline
- ✓ Competing processes and urgency
- ✓ Authorization, compliance, or client-specific constraints
- ✓ Willingness for the interview format being proposed

I like a simple traffic-light distinction in my notes. Green means aligned enough to proceed. Yellow means there is a known issue, but it is bounded and discussable. Red means stop now.

Do not downgrade a red flag because the profile looks attractive on paper. That is how teams end up interviewing fantasy candidates. I have made that mistake myself. The résumé remained lovely. The mismatch remained a mismatch.

Turning the conversation into a decision

A recruiter screen is only useful if it ends in a decision someone else can act on.

What works best for me is simple: debrief against the same four areas every time.

- fit for the role
- motivation to move
- communication and collaboration
- practical process alignment

Do not write everything you heard. Write what you learned.

Simple recruiter debrief structure

DECISION	WHAT IT MEANS	WHAT HAPPENS NEXT
Strong yes	Core fit is established with no meaningful flags	Move forward and tell interviewers what not to retest
Yes with flags	Worth progressing, but specific risks need testing	Move forward with targeted follow-up areas
Hold for clarification	Signal is incomplete or contradictory	Resolve one or two open questions before next step
No	Basics for this role are not there	Close respectfully and clearly

A strong yes means the candidate is directionally right across the board. A yes with flags means they are worth progressing, but specific risks need testing. A hold for clarification is useful when the story is polished but oddly content-light, or when one requirement is still too fuzzy. A no should be calm and specific: missing the core environment, weak evidence of hands-on depth, poor motivation for this move, or practical constraints that make the process pointless.

One thing that improves screen quality quickly is banning vague debrief language. “Not a fit” tells nobody anything. Say what is missing.

Calibration with the hiring manager matters here too. Many bad screens are not bad interviewing. They are bad intake translated into bad decisions. If a manager says they want “strong backend,” make them define it. Do they mean high-volume API work, meaning services that expose functionality to other systems? Do they mean distributed systems? Do they mean production ownership? Those are different screens.

The same goes for “good communication.” Good with whom, about what, and in what setting? Writing design proposals? Working with product? Handling disagreement in code review? Unless you pin that down, recruiters and engineers will imagine different people and then wonder why the pipeline feels chaotic.

What every debrief should leave behind

- ✓ Clear recommendation: strong yes, yes with flags, hold, or no
- ✓ Evidence across all four screening areas
- ✓ Specific risks translated into follow-up questions
- ✓ What engineering does not need to re-establish
- ✓ Any process or motivation issues that could derail later

Consistency beats theatrics. You do not need an elaborate competency matrix in seven shades of enterprise optimism. You need a stable frame and decent listening.

That broader shift is already visible in LinkedIn's Future of Recruiting 2025 report (https://business.linkedin.com/content/dam/lem/business/en/hire/resources/future-of-recruiting/staffing/future-of-recruiting-2025-staff.pdf), which argues that as AI handles more administrative work, recruiters should spend more time on judgment, communication, adaptability, and better assessment of skills. I think that is exactly right.

Conclusion

If you remember one thing from this book, let it be this: the recruiter screen is not the warm-up. It is where you establish whether there is enough substance to justify the rest of the process.

In practice, that means coming back to the same four questions every time. Is this person genuinely relevant to the role? Do they have a real reason to move? Can they communicate and collaborate in the way the team needs? And are the practical conditions aligned enough to proceed?

You do not need to become a pseudo-engineer to do this well. You need a clear frame, better follow-up questions, and the discipline to turn a conversation into a useful recommendation.

Across this series, I come back to the same idea in different forms: good hiring gets much easier when each step does its own job properly. In this book, the recruiter interview does that job by creating clarity early. It saves engineering time, improves candidate experience, and prevents a surprising amount of expensive confusion later.

And if it occasionally saves you from three extra rounds with someone who was never going to take the job, that is not just efficiency. That is self-care.

Glossary

- **AI (Artificial Intelligence)** — Software that performs tasks that usually need human judgment, such as generating text, code, or predictions.
- **API (Application Programming Interface)** — A defined way for one software system to interact with another.
- **Backend engineer** — An engineer who builds the server-side logic, data handling, and systems behind an application.
- **Code review** — A process where developers examine each other's code before it is merged into the product.
- **Distributed systems** — Software systems that run across multiple machines and must coordinate reliably.
- **Hybrid work** — A working model that combines remote work with required in-office time.
- **Kubernetes** — A platform used to deploy, run, and manage software applications across servers.
- **Large language model** — A type of AI model trained on large amounts of text and used for tasks like writing, summarizing, or answering questions.
- **Machine learning** — A branch of AI where software learns patterns from data to make predictions or decisions.
- **Machine learning engineer** — An engineer who builds, deploys, or maintains systems that use machine learning models.

- **On-call** — A work arrangement where an engineer is available outside normal hours to respond to urgent system issues.
- **Production** — The live environment where real users interact with software.
- **Quality assurance** — The work of testing software and improving confidence that it behaves correctly before release.
- **Retrieval pipeline** — A system that fetches relevant information for an AI application before it generates a response.
- **System design** — The work of planning how software components fit together to handle scale, reliability, and other requirements.
- **Total compensation** — The full value of a job offer, including salary, bonus, equity, and benefits, not just base pay.
- **Work authorization** — Legal permission for a candidate to work in a specific country or jurisdiction.

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.

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.