TECH RECRUITMENT 101

What Recruiters Should and Should Not Assess

PART OF THE *TECH RECRUITMENT 101* SERIES

What Recruiters Should and Should Not Assess

Knowing where your judgment adds value and where to hand the baton to engineers

CONTENTS

When I first started working in technical hiring, I saw the same problem again and again. Recruiters were told to assess technical candidates, but very few people drew a clear line around what that actually meant. Some recruiters stayed so far back that they became coordinators. Others stepped too far in and tried to judge code, architecture, or tooling depth they were never meant to own. Neither approach helped candidates, hiring managers, or engineers.

I wrote this because the middle ground is real, and it matters. If you know where your judgment adds value, you can run a much stronger screen. You can test career logic, motivation, ownership, communication, level fit, and practical constraints without pretending to be an engineer. You can also stop doing the things that weaken technical hiring, like keyword policing, stack memorization, and vague handoffs that force engineers to restart the evaluation from zero.

This topic feels more urgent now because polished materials are easier to produce than ever. Resumes are smoother. Profiles are sharper. AI can help candidates present themselves well. That does not make recruiter judgment less useful. It makes disciplined recruiter judgment more useful. We need better screens, cleaner evidence, and better partnership with engineers. That is why I wanted this topic to be part of the Tech Recruitment 101 series.

In the pages that follow, I focus on the real boundary. I cover where recruiter judgment actually matters, what you should assess with confidence, what you should not pretend to assess, how to build a clean handoff to engineers, how to handle messy profiles like career changers or fuzzy AI titles, and how to build credibility without technical theater. My goal is simple: help you leave this with a practical model you can use on your next req, your next screen, and your next debrief.

Where recruiter judgment actually matters

Technical hiring gets sloppy in two predictable ways. Recruiters either shrink their role until they become schedulers with LinkedIn access, or they try to act as junior engineers and judge depth they do not really own. Both mistakes cost teams good hires.

Your value is not a watered-down version of technical evaluation. It is a different evaluation.

The boundary is simple: judge what you can observe well and defend clearly. Do not guess at technical competence from surface signals, buzzwords, or a half-understood answer. A candidate should not fail a screen because they used a different term than the one in your intake notes.

That matters more now because polished stories are cheap. Skills-based hiring keeps growing, and employers are relying less on resumes alone as proof of fit. In TestGorilla's 2025 State of Skills-Based Hiring (https://www.testgorilla.com/skills-based-hiring/state-of-skills-based-hiring-2025/), 85% of employers reported using skills-based hiring, while resume use fell year over year. Candidates also increasingly use AI to write or refine resumes and other materials. In the Employ 2025 Job Seeker Nation Report (https://pages.lever.co/rs/659-JST-226/images/2025-Job-Seeker-Nation-Report.pdf), 52% said they use AI for writing or reviewing resumes. So the recruiter screen has to do more than reward fluency. It has to test coherence, specificity, and lived experience.

The evidence you own is often the evidence nobody else gathers early enough. You are usually the first person testing whether the candidate's story holds together. Why this move now? Why this role? Does the career path make sense? Can the candidate explain what they actually did, what the team owned, and where their contribution starts and ends?

A strong recruiter screen can assess six core things with confidence: career logic, motivation, communication, level alignment, practical constraints, and role understanding. Later in the process, you may also check ownership, compensation, and consistency, but those support the same core judgment rather than replace it.

Boundary map

RECRUITER EVALUATES	ENGINEER EVALUATES
Career logic and motivation	Code quality and debugging depth
Level fit and communication	System design and architecture
Constraints and role understanding	Technical tradeoffs in practice

The cost of crossing that boundary is real. When recruiters overreach, they create false negatives by rejecting capable people for the wrong reasons, and false positives by being impressed with stack names or rehearsed jargon. The candidate experience gets worse too. Nothing undermines trust faster than a screen where the recruiter asks technical questions they cannot evaluate. Going too far the other way is not better. If you judge nothing beyond availability and salary, you force engineers to start from zero.

LinkedIn's Future of Recruiting 2025 (https://business.linkedin.com/content/dam/lem/business/en/hire/resources/future-of-recruiting/future-of-recruiting-2025.pdf) found that 93% of talent professionals believe accurate skill assessment is crucial to quality of hire. I agree, with one

condition: accuracy depends on knowing which skills you can assess directly and which ones you should structure, then hand off.

After building recruiting teams a few times, I have found that the best recruiters are not wider. They are sharper. They know their lane well enough to make strong calls inside it, and clean handoffs outside it.

What you should assess with confidence

Your job is to test whether this person makes sense for this role, at this level, in this process, right now. That is not a smaller job. It is a different one.

Start with motivation and career logic. You are not looking for theater or a perfect ladder. You are looking for a next step that makes sense. “I’ve spent the last two years in internal tools, and I want a product role where user feedback is tighter” is useful. “I’m passionate about innovation” tells you nothing.

Assess ownership. By ownership, I mean the part of the work the candidate was personally responsible for driving, not just the part they were present for. Someone with ownership can usually explain the goal, their decisions, their tradeoffs, and what happened because of their work. Someone who participated may still have done solid work, but the shape is different: they implemented one piece, supported a lead, or handled a defined task inside a larger project. For example, “I scoped the migration, aligned the teams, made the rollout plan, and was accountable for the launch” signals ownership. “I built two endpoints that were part of the migration and reported to the tech lead” signals contribution, not end-to-end ownership. Ask narrow questions: what was the problem, what part was yours, who else was involved, what tradeoff did you make, what happened in the end. People who did the work usually answer in a straight line.

Assess communication. I do not mean polished presentation voice. I mean whether they can explain their work in plain language, with enough structure that another human can follow it. Engineers do not need to sound like conference speakers. They do need to make sense.

Assess role fit and level fit. Ask whether the scope of the candidate's past work matches the scope of the role. By level, I do not mean title prestige. I mean the size and complexity of problems they handled, how much independence they had, and how much other people relied on their judgment. A junior person usually works well on defined tasks with support. A mid-level person usually delivers whole pieces of work with normal guidance. A senior person usually handles more ambiguity, makes decisions across a broader scope, and influences how work gets done beyond their own tickets. This is also why titles can mislead. A "Senior Engineer" at a small company might be the most experienced person on a three-person team and mostly execute defined work, while a "Software Engineer II" at a larger company might own a critical service used by several teams and operate at a higher level of scope. Ask about system size, decision-making, cross-team coordination, and what they were trusted to own. That gives you a safer read on level than the title alone.

Then cover logistics and compensation early. If someone needs full remote and the role is hybrid, that matters. If their timeline, visa needs, or pay expectations do not fit, identify it early and cleanly. False hope is not candidate care.

Use a simple frame when you screen:

What I should validate before engineer review

DIMENSION	WHAT TO TEST
Motivation	Why this role now
Career logic	Do moves make sense
Ownership	What they drove themselves
Communication	Can they explain clearly
Role fit	Does scope match job
Level fit	Is seniority realistic
Logistics	Location, timing, process
Compensation	Are ranges aligned
Consistency	Resume, profile, call match

Consistency matters more than polish. AI can make a profile smoother, but it often flattens detail. Listen for specifics that connect naturally: names of systems, team constraints, sequence of decisions, what changed because of their work. Structured interviewing helps here. Cornell's 2025 brief notes growing use of question banks and rubrics, which is useful because structure

gives you observable evidence instead of vibes dressed as insight (Cornell ILR, New Developments in Structured Interviewing (https://www.ilr.cornell.edu/sites/default/files-d8/2025-01/new-developments-in-structured-interviewing_final_.pdf)).

Keep the screen conversational. Ask one grounded question at a time, then follow the thread. “Walk me through the project.” “What part was yours?” “What was hard?” “How did you decide?” If every answer stays broad, generic, and oddly frictionless, believe the pattern. Real work has edges.

If you can assess these things with confidence, you hand engineers something far more useful than a keyword match: a candidate with a credible story, clear context, and a real reason to be in the process.

What you should not pretend to assess

Your job is not to become a lightweight engineer with a scheduling link.

Candidates can tell when a recruiter is bluffing technical depth. Engineers can too. Once that happens, your screen loses value.

The line is simple. You can clarify what a candidate worked on, how much ownership they had, how they communicate, what problems they tend to solve, and whether their background maps to the role. You should not decide whether their code was elegant, their architecture was sound, or their approach was technically correct.

That matters even more in messy areas like AI, infrastructure, data, and security, where titles are mushy and the same label can describe very different work. Your job is to clarify what the company means and what the candidate actually did. Your job is not to judge engineering design choices like which database they used, how they measured model quality, or how the system stored data temporarily to improve speed.

Here is where recruiters usually overreach.

Do not judge code quality from a resume or verbal summary. Ask what they owned, who they worked with, what changed because of their work, and how success was measured.

Do not judge architecture tradeoffs. If a backend engineer says they split a monolith into services, meaning one large application broken into smaller services, or changed caching strategy, your role is to understand the problem they were solving, not whether the choice was correct.

Do not judge framework depth unless the role requires a very narrow tool match and the hiring manager has given you exact screening guidance. “Used React” tells you little. “Led migration from Angular to React while mentoring junior engineers and partnering with design” tells you something useful.

Do not judge debugging skill from confidence. Ask for a concrete example, then hand that evidence to engineers, who can tell whether the reasoning holds up.

Do not judge research depth in areas you do not practice. Capture what they did and why it mattered. Do not decide whether the method was robust.

Where your judgment stops

YOU ASSESS	ENGINEERS ASSESS
Scope of work	Code quality
Ownership and influence	Architecture tradeoffs
Clarity and credibility	Framework depth
Motivation and fit	Debugging ability
Transferable signals	Technical rigor in niche areas

The worst trap is keyword policing. LinkedIn's Skills Signal Report 2025 (https://economicgraph.linkedin.com/content/dam/me/business/en-us/talent-solutions/resources/pdfs/the-skills-signal-report-2025.pdf) found that workers matched by skills rather than titles qualify for more than three times as many roles. If you reject a strong platform engineer because they said "event-driven systems" instead of naming the exact broker you expected, you are not protecting quality. You are shrinking the pool for no good reason.

The second trap is false confidence from memorizing stacks. Familiar vocabulary helps you frame questions. It does not grant authority to score the answer.

The third trap is punishing simple explanations. Strong people often explain complex work plainly because they understand it. Weak people often hide behind jargon because they hope you do not ask the next question. Reward coherence, sequence, and specificity.

This is also why resume review must stay in its lane. CoderPad's 2026 State of Tech Hiring (https://coderpad.io/survey-reports/coderpad-state-of-tech-hiring-2026/) found that 69% use resume review for technical hiring, but only 16% think it predicts on-the-job performance. The better predictors were technical discussion and live evaluation, both led by people who can judge the substance.

A good recruiter screen ends with notes like these: built internal tools used by a support team; owned rollout across several functions; explains tradeoffs clearly; moved from data analyst work into analytics engineering; motivation for this domain is strong; needs technical panel to test SQL depth and data modeling.

You do not need to become timid. You need to become precise.

How to build a clean handoff to engineers

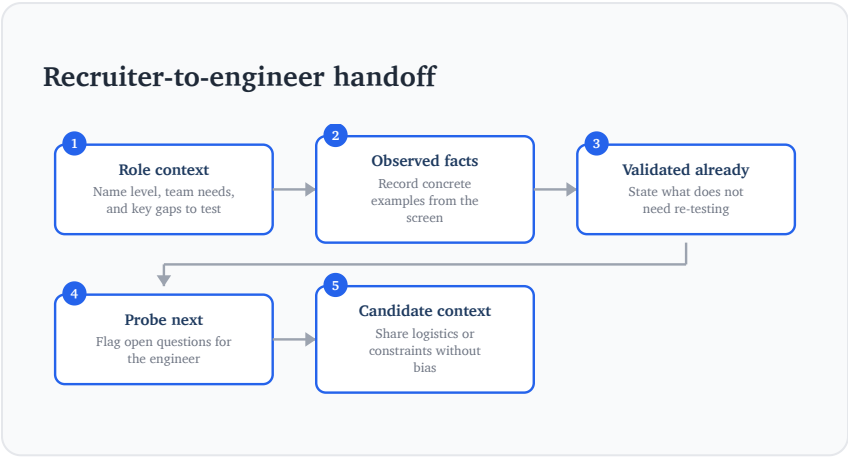
Once you know your boundary, the next job is simple: make the technical interview sharper, not redundant.

A recruiter screen should gather useful evidence, package it clearly, and hand engineers a better starting point. The simplest rule is this: write what you learned, not the verdict you wish you could give.

“Strong technically” is not useful. “Led a service migration, explained tradeoffs between speed and risk, and gave clear examples of cross-team coordination” is useful.

Separate facts from interpretation every time. Facts are what the candidate said or did. Interpretation is what you think it may mean. Both can help, but they should never blur together.

Here is a format I like after building recruiting teams a few times:



Start with role context. Engineers should know what problem they are solving in the interview. If you skip that, interviewers fill the gap with their own assumptions.

Then list observed facts in short bullets. Keep each bullet concrete. Good notes often include project scope, ownership, decision-making, communication clarity, and signs of how the candidate worked with others.

Next, name what you already validated. If you confirmed compensation range, work authorization, location constraints, motivation, and the candidate's ability to explain past work clearly, say so. Engineers do not need to spend fifteen minutes rediscovering facts you already have.

Then flag what to probe. This is where recruiter judgment adds value without overreach.

For example:

- Validated: explained ownership of a production migration from planning through rollout
- Validated: described team context and stakeholders clearly
- Probe: depth of architecture choices
- Probe: personal contribution versus team contribution
- Probe: debugging approach under failure conditions

Here is what a finished note can look like:

- Role context: Senior backend engineer for a product team rebuilding internal billing workflows; panel should test system design depth, debugging under production pressure, and level fit for cross-team ownership.
- Observed facts: Candidate spent four years at a mid-size SaaS company on backend services for payments and invoicing. Described leading a migration from a legacy billing service to a new service owned by their team. Said they scoped rollout phases, coordinated with product and support, and handled launch communication. Explained one incident clearly, including what failed, how the team narrowed the issue, and what changed after.
- Validated already: motivation for this move is coherent; compensation range aligned; hybrid expectation aligned; can explain past work in a structured way; has experience working across functions, not only within engineering.
- Probe next: depth of system design decisions in the billing migration; how much architecture direction was theirs versus the staff engineer's; strength of debugging in unfamiliar failure modes; whether scope matches senior expectations in this environment.
- Candidate context: needs four weeks' notice; no visa support needed.

Be careful with candidate context. Share facts that affect the interview. Do not preload the panel with your conclusion. "Career move from backend to platform work" helps. "I think she is a hidden gem" does not.

Structured notes also improve calibration. Cornell ILR's 2025 brief on structured interviewing (https://www.ilr.cornell.edu/sites/default/files-d8/2025-01/new-developments-in-structured-interviewing_final_.pdf) points to the value of question banks, rubrics, and clearer evaluation structure. The same principle applies to handoffs. If every recruiter writes notes differently, engineers cannot compare candidates fairly, and you cannot spot where your screen is helping or hurting.

Use a short review loop with hiring managers and interviewers. Look at misses together. Which candidates did recruiters send forward with confidence that engineers rejected quickly? Which candidates did engineers love that recruiters nearly screened out? That is where your boundary map gets better.

Clean handoff checklist

- ✓ State role context and what this interview should test
- ✓ Write observed facts, not fuzzy labels
- ✓ Separate interpretation from evidence
- ✓ Mark what is already validated
- ✓ Flag 2 to 4 areas to probe next
- ✓ Share context without steering the outcome
- ✓ Review misses with engineers and adjust

A good handoff helps the next interviewer start closer to the truth. That is real recruiter value.

Boundary map for messy cases

Edge cases expose weak recruiting faster than easy reqs do. When the profile does not line up neatly with the title, many recruiters drift into guesswork. Do not. Collect evidence about fit, scope, and transferability, then hand technical judgment to the people equipped to make it.

That matters because hiring is moving away from title matching and toward evidence of skills. In 2025, 85% of employers reported using skills-based hiring, while resume use dropped year over year. Candidates matched by skills rather than titles also qualify for far more roles, which is exactly why messy profiles deserve better questions, not faster rejection (TestGorilla, State of Skills-Based Hiring 2025 (https://www.testgorilla.com/skills-based-hiring/state-of-skills-based-hiring-2025), LinkedIn, The Skills Signal Report 2025 (https://economicgraph.linkedin.com/content/dam/me/business/en-us/talent-solutions/resources/pdfs/the-skills-signal-report-2025.pdf)).

A career changer moving between stacks is the clearest example. If a backend engineer worked in Java and now wants a role in Python, do not ask whether they are secretly already a senior Python engineer. Ask what they built, how complex the systems were, how they learned new tools before, and what part of the move is familiar versus new. You are assessing ramp risk and evidence of transfer, not certifying language depth.

The same rule helps with engineers returning after a break. A gap does not tell you much by itself. Ask what they worked on before the break, what has changed in their target area, and what they have done to get current. A strong answer sounds specific. A weak answer stays vague and hopeful.

Titles are another trap. “Lead,” “Staff,” and “Principal” can describe very different jobs across companies. Strip the title off and ask about scope: system size, decision rights, cross-functional work, and who relied on their judgment. Scope travels better than labels.

Senior candidates create a different problem. Some can talk at a high level for forty minutes and still leave you unsure whether they ever executed anything difficult themselves. Pin broad claims to concrete work. If every answer floats upward into vision, you may be looking at borrowed altitude.

AI roles make this blurrier because titles are still unstable. When a candidate says they built AI products, verify five things: the scope of the work, the production environment, the team setup, the business context, and the role they want now. Did they prototype internal tools, tune prompts for a workflow, evaluate model output, ship features into production, or train models from scratch? Those are not the same job. Your task is to map the work, not judge whether their model selection was technically correct.

For messy profiles, verify

- ✓ What they actually built
- ✓ How complex the setting was
- ✓ What they owned directly
- ✓ What changed because of them
- ✓ How target role matches past work

The worst mistake in messy cases is trying to rescue a vague process by making technical calls yourself. If the hiring manager cannot explain level, scope, or must-have depth, stop and force clarity upstream. I have seen recruiters improvise downstream and accidentally screen out strong people or wave through confident ones with thin execution.

Your boundary is simple under pressure: evaluate fit, evidence, communication, ownership, and transferability. Defer code-level depth, architecture quality, and technical correctness to engineers.

A practical operating model for recruiter credibility

Credibility does not come from sounding like an engineer. It comes from being precise, consistent, and clear about what you assess and what you hand to engineers.

Your lane starts before the first screen. Push for role clarity before kickoff. Ask what problems this person will solve in the first six to twelve months, which skills are required on day one, and which can be learned. If the hiring manager cannot explain the difference, you do not have a recruiting problem yet. You have a scope problem.

In the screen, ask for evidence, not performance. Replace “How strong are you in Python?” with “Tell me about a recent problem you solved with Python. What was hard about it? What did you own?”

Keep notes that another person can use. Write what the candidate did, why they did it, what changed because of their work, and where your confidence is high or low. “Good communicator” is not a note. “Explained a production incident clearly, named tradeoffs, and answered follow-up questions directly” is a note.

Use a shared rubric. Cornell ILR’s 2025 brief on structured interviewing (https://www.ilr.cornell.edu/sites/default/files-d8/2025-01/new-developments-in-structured-interviewing_final_.pdf) points to the value of

question banks and scoring rubrics. A simple rubric keeps the screen honest. It also makes your handoff stronger because engineers receive organized evidence, not vibes in business casual.

Recruiter credibility in practice

DO ASSESS	DEFER TO ENGINEERS
Motivation for this role	Code quality and correctness
Communication and clarity	System design depth
Ownership and scope	Technical tradeoff quality
Career context and level fit	Framework-specific expertise
Evidence of transferability	Edge-case technical judgment

Say the boundary out loud. “I can assess role fit, communication, ownership, and how you think about your work. I do not assess code-level depth in this screen. The engineering team will do that.” This does not make you look weak. It makes you look rigorous.

Candidates trust rigor. Employ’s 2025 Job Seeker Nation Report (https://pages.lever.co/rs/659-JST-226/images/2025-Job-Seeker-Nation-Report.pdf) found that 58% trust HR staff more than AI to guide them

through the interview process. Your value is not technical theater. Your value is a cleaner signal.

Use this test before every screen: am I trying to sound technical, or am I making the hiring decision clearer? After building recruiting teams a few times, I can tell you which one earns trust.

The main shift I want you to take from this is simple. You do not become more credible by stretching your judgment past its limit. You become more credible by drawing the line clearly and working that line well. If you can assess motivation, ownership, communication, level fit, constraints, and transferability with discipline, then write clean handoffs that tell engineers exactly what to probe next, you make the whole hiring process better. That is true for straightforward reqs and even more true for messy ones, where title matching breaks down and vague technical theater does real damage.

So the next time you open a req, do two things first. Get clear on what belongs in your lane and what belongs with engineers. Then build your screen around observable evidence, not borrowed confidence. Ask what they built, what they owned, why the move makes sense, and where the open technical questions still are. That one change will sharpen your screens fast.

If this topic was useful, the other books in the Tech Recruitment 101 series go deeper into the rest of the recruiting craft around it. They are built the same way: practical, specific, and meant to help you do the job with more clarity and less noise.

Glossary

- **AI (Artificial Intelligence)** — Technology used to generate, analyze, or improve content and software-related work. In hiring, it often comes up when candidates use AI tools for resumes or when roles involve building AI-powered products.
- **AI titles** — Job titles that reference artificial intelligence work but may mean very different things across companies. Recruiters should clarify the actual work behind the title instead of assuming the scope.
- **Analytics engineering** — Work that sits between data analysis and software engineering, usually focused on preparing, modeling, and organizing data so it can be used reliably by the business. It often involves building data pipelines and datasets for reporting or analysis.
- **Angular** — A frontend web framework used to build user interfaces and applications. Seeing Angular on a resume tells you the candidate has used that tool, but not automatically how deeply.
- **Architecture** — The high-level structure of a software system and how its components fit together. Recruiters usually should not judge whether an architecture was “right,” but can ask what problem it was meant to solve.
- **Architecture tradeoffs** — Decisions engineers make when balancing competing needs such as speed, reliability, complexity, or cost in system design. These are usually best assessed by engineers rather than recruiters.

- **Backend engineer** — An engineer who works on the server-side parts of software, such as business logic, APIs, databases, and services behind an application. They typically build and maintain the systems users do not directly see.
- **Billing workflows** — The processes and systems a company uses to manage charging, invoicing, and payment-related operations. In a tech context, this often involves internal tools and backend services.
- **Broker** — A software component that routes or manages messages or events between systems. It is often part of event-driven architecture.
- **Caching** — Temporarily storing data so it can be retrieved faster later. Engineers use caching to improve application speed and reduce repeated work.
- **Code quality** — A general measure of how clear, maintainable, reliable, and effective code is. It is an engineering assessment area, not something recruiters should infer from buzzwords alone.
- **Conference speakers** — People who present at industry events, often used here as a contrast to everyday engineers. The point is that engineers do not need polished public-speaking style to communicate their work clearly.
- **Cross-functional** — Work that involves collaboration across multiple functions such as engineering, product, support, or design. It signals that a candidate worked beyond a single team silo.

- **Cross-team coordination** — Working across multiple engineering or business teams to align plans, dependencies, and delivery. This can be a useful signal of scope and seniority.
- **Data** — A broad technical area involving the collection, storage, transformation, and use of information in systems. In hiring, "data" roles can range widely, so recruiters should clarify the actual responsibilities.
- **Data analyst** — A professional who works with data to find patterns, produce insights, and support decision-making. Compared with analytics engineering, the role is usually more focused on analysis than building data systems.
- **Data modeling** — The practice of organizing data into structures that make it usable, consistent, and efficient for analysis or applications. Engineers or data specialists usually assess depth here.
- **Database** — A system for storing and retrieving structured information. Different databases are suited to different technical needs, but recruiters typically only need to understand the context of use.
- **Debugging** — The process of finding, understanding, and fixing problems in software. Recruiters can ask for examples, but engineers are usually the right people to judge depth of debugging skill.
- **Edge-case technical judgment** — Evaluation of unusual or highly specialized technical situations that require deep subject-matter expertise. This is typically outside a recruiter's scope.

- **End-to-end ownership** — Responsibility for driving work from initial planning through execution and outcome, rather than only contributing one part. It is a strong signal of scope and accountability.
- **Engineer** — A technical professional who designs, builds, tests, or maintains software systems. In this book, engineers are the people who should assess technical depth.
- **Event-driven systems** — Systems where actions are triggered by events, such as a user action or a system update, rather than by a fixed sequence alone. They are common in distributed software architectures.
- **Framework** — A reusable software foundation that gives developers a standard structure for building applications. Examples in the text include Angular and React.
- **Framework depth** — How well someone understands and can work within a specific software framework. This is more than having used the tool once; it implies practical fluency.
- **Frontend** — The user-facing part of a software application, such as screens, buttons, and interactions in a web app. Frontend work is often built with frameworks like Angular or React.
- **HR (Human Resources)** — The business function responsible for people processes such as hiring, onboarding, and employee support. In this text, it appears in the context of candidate trust during interviews.
- **Hybrid** — A work arrangement where employees split time between remote work and working in a physical office. It is a

practical hiring constraint that often needs to be confirmed early.

- **Infrastructure** — The underlying systems and services that support software, such as servers, networks, cloud resources, and deployment foundations. Infrastructure roles can vary a lot, so titles alone may be misleading.
- **Internal tools** — Software built for employees inside a company rather than for external customers. These tools help teams do work such as support, operations, or billing.
- **Java** — A programming language commonly used for backend and enterprise software development. Recruiters may see it as a stack component, but should focus on what the candidate built with it.
- **Jargon** — Specialized technical language used within a field. In hiring screens, jargon alone is a weak signal unless the candidate can explain the underlying work clearly.
- **Junior engineer** — An early-career engineer who usually works on more defined tasks with guidance from others. The term reflects level and scope rather than just years of experience.
- **Legacy** — Older software or systems that are still in use but may be harder to maintain or extend. A “legacy billing service” means an existing older system being replaced or updated.
- **LinkedIn** — A professional networking platform often used in recruiting and talent research. In this text it also appears as the source of hiring-related reports.

- **Live evaluation** — A real-time technical assessment, often involving discussion or problem-solving during an interview. It is presented in the text as a stronger predictor than resume review alone.
- **Mid-level** — A career stage between junior and senior where someone can usually deliver complete pieces of work with normal guidance. It reflects expected scope, independence, and consistency.
- **Migration** — Moving software, systems, or data from one setup to another, such as from an old service to a new one. Migrations often involve planning, risk management, and rollout coordination.
- **Model output** — The results produced by an AI or machine learning model. Evaluating whether those results are useful or accurate depends on the context and the goals of the system.
- **Model quality** — How well an AI or machine learning model performs according to the chosen success measures. This can include accuracy, usefulness, or reliability depending on the task.
- **Monolith** — A single large application where many functions are built and deployed together. Teams may later split a monolith into smaller services as systems grow.
- **Panel** — A group interview or interview stage involving multiple interviewers, often used to assess different aspects of a candidate. In technical hiring, panels may test design, debugging, or level fit.

- **Platform engineer** — An engineer who builds and maintains the internal systems, tooling, or infrastructure that other engineers use. The role often focuses on scalability, reliability, and developer productivity.
- **Principal** — A senior technical level or title that often implies broad influence and high-level decision-making. Its meaning varies a lot across companies, so recruiters should validate scope rather than trust the label.
- **Production** — A live software environment used by real customers or real business operations. Work done in production usually carries higher reliability and business impact expectations.
- **Product role** — A role focused on building software or features for end users or customers, often with tighter feedback loops from usage. In this text, it contrasts with internal tools work.
- **Profile** — A candidate's professional summary or public career representation, often on platforms like LinkedIn. In technical hiring, profiles may be polished but still need validation through conversation.
- **Prototype** — An early or simplified version of a tool or product built to test an idea quickly. Prototype work is different from building and supporting something in production.
- **Python** — A programming language widely used in backend development, automation, data, and AI work. Recruiters

should focus less on the name itself and more on what the candidate accomplished with it.

- **Question banks** — Collections of prewritten interview questions used to create more structured and consistent interviews. They help recruiters and interviewers gather comparable evidence.
- **React** — A frontend JavaScript library used to build user interfaces. It is a common tool in web development but the tool name alone does not show depth.
- **Req** — Short for job requisition, meaning an approved open role a company is hiring for. In technical recruiting, each req may have different scope, level, and must-have skills.
- **Resume review** — The process of evaluating a candidate based on their resume before interviews. The text notes that it is widely used but not seen as a strong predictor of job performance on its own.
- **Rubric** — A structured scoring guide used to evaluate candidates consistently against defined criteria. Rubrics help make interview decisions more evidence-based.
- **SaaS (Software as a Service)** — Software delivered over the internet as an ongoing service rather than installed locally once. Companies that build SaaS products often run backend services, billing systems, and user-facing applications.
- **Scheduling link** — A tool or URL used to let candidates book interview times automatically. The text uses it as shorthand for admin coordination rather than technical evaluation.

- **Security** — The practice of protecting systems, data, and software from misuse, attack, or unauthorized access. Security roles and responsibilities can differ widely by company.
- **Senior** — A more experienced level where someone is expected to handle greater ambiguity, broader scope, and stronger decision-making influence. It is better assessed through actual work scope than title alone.
- **Senior Engineer** — An engineer at a more advanced level who usually works with greater independence, wider system scope, and more influence on how work gets done. The exact expectations vary by company.
- **Service** — A software component or application that performs a specific function, often as part of a larger system. Services can be internal, customer-facing, or part of a distributed architecture.
- **Site Reliability Engineer** — A role name abbreviated as SRE in many companies, focused on keeping systems reliable, scalable, and available. This exact abbreviation does not appear in the text, but the reliability-focused engineering domain is adjacent to the infrastructure topics discussed.
- **Skills-based hiring** — A hiring approach that emphasizes demonstrated skills and evidence of ability over titles, pedigree, or resume polish alone. It is presented in the text as a growing trend in recruiting.
- **Software Engineer II** — A company-specific engineering title that usually signals an intermediate level. As the text notes,

title alone can be misleading without understanding scope and ownership.

- **SQL** — A language used to query and manage data in relational databases. In this text, it appears as a technical area engineers should test directly for depth.
- **Stack** — The set of technologies, languages, frameworks, and tools used to build software. Recruiters often hear about stack names, but should avoid treating them as a full measure of capability.
- **Staff engineer** — A senior individual contributor role that usually involves broad technical influence, complex problem-solving, and guidance across teams. The exact scope varies by company.
- **Structured interviewing** — An interview approach that uses predefined questions, scoring criteria, and consistent evaluation methods. It helps reduce vague judgments and improve fairness.
- **Support team** — A team that helps users or internal stakeholders resolve issues and keep operations running smoothly. Technical candidates may build tools used by support teams.
- **System design** — Planning how a software system should be structured, including components, data flow, and tradeoffs. It is a core technical interview area usually owned by engineers.
- **System size** — The scale of the software environment a candidate worked in, such as how large, complex, or widely

used it was. This can help recruiters judge scope and level without judging code itself.

- **Technical interview** — An interview stage designed to assess engineering or technical competence more directly. Recruiters can improve it by handing off clear context and evidence.
- **Technical panel** — A group interview conducted by technical interviewers to assess skills such as SQL, system design, or debugging. It is often used after the recruiter screen.
- **Technical rigor** — The depth, care, and correctness expected in technical work or reasoning. In niche areas, this is best judged by engineers with domain expertise.
- **Tooling** — The software tools and development environment engineers use to build, test, deploy, or maintain systems. Recruiters should avoid overestimating capability based on tool names alone.
- **Tradeoff** — A decision where improving one thing usually means giving up something else, such as speed versus risk or flexibility versus simplicity. Tradeoffs are central to engineering judgment.
- **Transferability** — The likelihood that skills or experience from one technical environment will carry over effectively to another. Recruiters can assess this by exploring what was built, owned, and learned.
- **Visa** — Government authorization that may allow someone to work in a particular country. In technical hiring, visa needs are an important practical constraint to confirm early.

- **Workflow** — A sequence of tasks or steps used to complete a process, often supported by software. In the AI example, a workflow is the business or operational process the tool is helping with.

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 1.1 — 6 April 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.