TECH RECRUITMENT 101

TECH ROLES EXPLAINED

A recruiter's field guide to what technical roles actually do

PART OF THE *TECH RECRUITMENT 101* SERIES

Tech Roles Explained

A recruiter's field guide to what technical roles actually do

CONTENTS

Tech Roles Explained

Introduction

I wrote this book for recruiters who already know how recruiting works but want a clearer mental model for technical roles. In tech hiring, titles often sound precise right up until you compare them across companies. Then everything goes a bit sideways.

I have been on both sides of this table for years, and one pattern keeps repeating: recruiters get blamed for "not understanding tech" when the real problem is usually that the role itself was defined loosely. A hiring manager says "backend," the résumé says "full-stack," the profile lists Python, AWS, and SQL, and suddenly everyone is pretending the title explained more than it did.

What I have found is that you do not need to become an engineer to recruit technical roles well. You do need a practical way to translate titles into actual work: what the person owns, what skills matter, what stacks are common, and what still needs to be clarified before you source.

That is what this book does. I will walk you through the core role families you will see most often, where the boundaries usually sit, where titles get messy, and how to build simple role cards you can actually use in intake, sourcing, and screening. If you are reading other books in this series, this one is the field guide you keep open while doing the real work.

Why tech titles confuse recruiters

The first thing I want you to stop doing is taking technical titles at face value. They look precise. They are not.

In software hiring, titles are shaped by company stage, team habits, architecture, and whoever wrote the job description. A backend engineer at one company may own application logic and application programming interfaces (APIs), the interfaces systems use to talk to each other. At another, the same title may include cloud infrastructure, internal tooling, or data pipeline work. I have seen similar work labeled backend, platform, DevOps, infrastructure, and full-stack, sometimes inside the same company.

The stack does not solve this for you. Common tools overlap heavily across roles. In the Stack Overflow Developer Survey 2024 (https://survey.stackoverflow.co/2024/), JavaScript, Python, SQL, TypeScript, PostgreSQL, and Amazon Web Services (AWS) all show up widely across the profession. That is exactly why you cannot see "Python" or "AWS" on a profile and conclude what kind of engineer you have.

Full-stack is the classic trap. Sometimes it means genuine breadth across frontend and backend. Often it means frontend-heavy with enough backend to ship, or backend-heavy with enough frontend to unblock themselves. Useful title, vague signal.

So my rule is simple: do not recruit from titles alone. Recruit from the work.

That means turning every role into a short reference card before you source.

Role reference card

TITLE

The market-facing name of the role

WHAT THEY DO

Day-to-day work and core ownership

KEY SKILLS

- Technical strengths
- Depth areas
- Scope of responsibility

COMMON STACKS

Typical tools, languages, frameworks, cloud

CLARIFY BEFORE SOURCING

What the title hides and what the hiring manager really needs

If you build this card early, title noise drops fast. "Backend engineer" becomes "owns customer-facing APIs, strong in Java and PostgreSQL, not an infrastructure role." "Full-stack engineer" becomes "React-heavy product engineer who can work in Node.js when needed." That is searchable. The title alone is not.

The most useful field is usually "clarify before sourcing." Ask:

- What does this person spend most of the week doing?
- What do they own directly?
- What is required versus merely familiar?
- If they came from a differently titled team, would they still fit?

Those answers tell you more than another round of title polishing.

Artificial intelligence has added another layer of title noise. Yes, AI engineer is now a real hiring category, and demand has grown meaningfully, as LinkedIn Economic Graph's 2025 AI labor market update (https://economicgraph.linkedin.com/content/dam/me/economicgraph/en-us/PDF/ai-labor-market-update-header-sept-2025.pdf) shows. But AI tool use is also broad across ordinary software work. The Stack Overflow Developer Survey 2025 AI section (https://survey.stackoverflow.co/2025/ai/) found very high adoption of AI tools alongside much lower trust in their output. So when you see AI on a résumé, do not assume the person builds models or machine

learning infrastructure. They may simply be doing standard product engineering with AI-assisted tools, like much of the industry.

That distinction matters. A Machine Learning Engineer (ML engineer) usually points to model deployment, feature pipelines, evaluation, or machine learning operations. An AI engineer may mean applied product work using large language models, retrieval, orchestration, and evaluation. Or it may mean a software engineer on a team that added an AI feature and updated the title. Again: get the work, not the label.

Product engineers: frontend, backend, full-stack, and mobile

These are the roles recruiters see most often, and they are easier to separate if you focus on where the engineer spends most of their time, what they own, and what trade-offs they are trusted to make.

Frontend engineers work on the user-facing application layer: what runs in the browser, how interfaces behave, how data moves through the app, and how users experience speed, clarity, and reliability. Good frontend work is not just visual polish. It includes accessibility, component design, rendering performance, testing, state management, and browser quirks, because browsers still refuse to behave like mature adults.

A strong frontend profile usually shows depth in one or more web frameworks such as React, Angular, or Vue, along with JavaScript or TypeScript, Cascading Style Sheets (CSS), and practical testing. TypeScript is an especially useful signal for modern web product work, and GitHub's Octoverse 2025 (https://github.blog/news-insights/octoverse/octoverse-a-new-developer-joins-github-every-second-as-ai-leads-typescript-to-1/) reflects how central it has become. But "used React" is not the same as frontend depth. Plenty of backend-heavy engineers have touched a settings page and then allowed the résumé to get ambitious.

Backend engineers work on server-side application logic: APIs, business rules, data models, integrations, background jobs, authentication, permissions, and the systems that make product features function. Common languages include Java, C#, Python, Go, Ruby, Kotlin, and Node.js, usually alongside databases, caches, queues, and cloud services.

The main hiring mistake here is confusing backend with infrastructure. They overlap, but they are not the same. A backend engineer may know AWS and deployment basics, but their core work is usually product functionality, data flow, and service behavior. If the role is really about cloud architecture, Kubernetes, Continuous Integration and Continuous Delivery (CI/CD), or infrastructure automation, you are drifting into platform, DevOps, or Site Reliability Engineering (SRE).

Full-stack engineers sit in the middle and often hide the real requirement. In the strongest version, full-stack means someone can deliver a feature end to end across frontend and backend. In the looser version, it means one side is clearly stronger and the other side is good enough. Neither is wrong, but they are different searches.

So I always ask where the center of gravity is. If someone has mostly built services, data models, and APIs with some React on top, I treat them as backend-leaning full-stack. If they have mostly built user-facing applications and can also create endpoints and

wire up persistence, I treat them as frontend-leaning full-stack. This sounds picky until you send the wrong profile and hear, "Good engineer, wrong shape."

Mobile is its own family. Cross-platform frameworks matter, but iOS and Android still matter too. Mobile engineers deal with app lifecycle, offline behavior, push notifications, release processes, device constraints, software development kits (SDKs), and platform-specific user interface patterns that web teams rarely think about until they have to.

Native mobile usually means Swift or Objective-C for iOS, and Kotlin or Java for Android. Cross-platform often means React Native or Flutter. The key recruiting point is that cross-platform does not erase platform depth. Teams still need people who understand operating system behavior, real-device debugging, and the differences between Apple and Android ecosystems. The State of React Native survey (https://survey.stateofreactnative.com/) is useful because it reflects current mobile team realities rather than generic web habits.

Practical role card for product engineers

ROLE	WHAT THEY USUALLY OWN	USEFUL SCREENING CLUES
Frontend	Browser app, interface behavior, user experience quality	Framework depth, TypeScript, accessibility, performance
Backend	APIs, business logic, data, integrations	Service design, databases, auth, scalability
Full-stack	Feature delivery across UI and server	Which side is stronger, end-to-end ownership
Mobile	iOS, Android, device app behavior	Native vs cross-platform, store releases, device constraints

When intake is fuzzy, a fast filter helps: **Which part of the product breaks if this person leaves?** The browser app, the server-side feature logic, both, or the mobile app. That answer is usually more useful than the title.

Quality, reliability, and the systems behind the product

This area gets messy because the jobs are adjacent, the tools overlap, and titles are often sloppy. I have seen companies use "DevOps" for three different jobs and "QA" for anyone vaguely near testing.

The cleanest way to sort these roles is by asking where they sit in the software lifecycle. One group focuses on finding defects before release. Another focuses on how code gets built, tested, and shipped. A third focuses on what happens after release, when users and traffic start doing what they do best.

What to clarify before you source

AREA	CORE QUESTION	TYPICAL OWNERSHIP
QA / testing	Does the product work?	Test coverage, defects, release confidence
DevOps / platform	Can we build and ship it reliably?	CI/CD, environments, infrastructure
SRE	Can it stay healthy in production?	Reliability, incidents, observability, service levels

Quality Assurance (QA) is about validating that the product behaves as expected. That can include manual testing, exploratory testing, regression testing, and acceptance testing. Manual QA still matters, especially in fast-moving products with a lot of user interface detail or messy edge cases. Automated tests are powerful, but they do not replace human judgment.

Test automation is different. Here you are hiring someone to write code that verifies the system automatically. Titles vary: Software Development Engineer in Test (SDET), test automation engineer, QA automation engineer. The plain-English version is that they build testing systems, not just test cases. They may work with browser automation tools such as Playwright or Selenium, API testing frameworks, integration tests, and test data setup. JetBrains' 2024 State of the Developer Ecosystem (https://www.jetbrains.com/lp/devecosystem-2024/) is useful here because it notes how tools like Playwright and Selenium have made more complex testing scenarios practical.

So before you open a testing role, pin down whether the team needs someone to execute tests, someone to design and maintain automation, or product engineers who are simply expected to own testing themselves.

DevOps is even noisier. Strictly speaking, DevOps began as a way of working: developers and operations collaborating to ship software faster and more safely. In practice, companies often use

"DevOps engineer" for people handling cloud infrastructure, CI/CD, infrastructure as code, deployment pipelines, secret management, and environment setup.

That shorthand is common, but it hides important differences. If a hiring manager asks for DevOps, I want to know whether they mean Terraform, Kubernetes, Amazon Web Services foundations, pipeline work, or internal tooling for developers. Those are related problems, not the same search.

Platform engineering is often what older job descriptions would have called DevOps with clearer boundaries. Platform engineers build the internal systems other engineers use to deploy, run, and operate software: self-service environments, paved-road infrastructure, templates, guardrails, and internal developer platforms. The point is not just to manage cloud resources. It is to make the rest of engineering faster and less chaotic. That is one reason the 2024 DORA / State of DevOps report (https://dora.dev/report/2024) treats platform engineering as a meaningful productivity lever, and why the Cloud Native Computing Foundation (CNCF) introduced a platform engineering certification in 2025 (https://www.cncf.io/blog/2025/06/15/introducing-the-certified-cloud-native-platform-engineering-associate-cnpa-community-driven-certification-for-platform-engineers/).

Site Reliability Engineers (SREs) sit closer to live service health. They focus on availability, latency, observability, incident response, capacity, error budgets, and Service Level Objectives (SLOs), targets for how reliable a service should be. Some SRE

roles are software-heavy and expect coding. Others are more operations-heavy. The key distinction is whether the team is mainly enabling delivery or mainly keeping production healthy.

Cloud engineering can overlap with all of the above, which is why the title alone tells you very little. Clarify whether the role is about cloud foundations, migrations, networking, account structure, security, or support for application teams.

Data, AI, security, and adjacent specialist roles

Specialist roles are where loose title matching starts wasting serious time. The tools overlap, the keywords overlap, and the actual work often does not.

The simplest way to separate these roles is to ask: **What system or decision did this person own?**

For data roles, I start with the split between analysis and infrastructure. A data analyst is usually closest to dashboards, metrics, business questions, experiments, and stakeholder requests. They turn raw data into decisions. Useful signals are SQL, visualization tools, experimentation literacy, and the ability to explain numbers plainly.

A data engineer sits further upstream. Their job is to make data reliable and usable. That means pipelines, Extract Transform Load or Extract Load Transform (ETL/ELT), orchestration tools, data warehouses, schema design, and data quality. They may use the same SQL and Python as analysts, but the emphasis is different. Analysts consume and shape data for decisions. Data engineers build the plumbing.

Data scientist is a broad title, so you need to pin it down early. In one company it means advanced analytics and experimentation. In another it means predictive modeling or research-heavy

machine learning. The U.S. Bureau of Labor Statistics Occupational Outlook Handbook for Data Scientists (https://www.bls.gov/ooh/math/data-scientists.htm) is a useful reminder that this is a major hiring family, but the label itself still covers too much ground. In screening, I care less about prestige words and more about methods used, production exposure, and what decisions or models they actually owned.

Machine Learning Engineers (ML engineers) are usually easier to distinguish once you stop treating them as renamed data scientists. They sit closer to production software: model serving, feature pipelines, evaluation, monitoring, versioning, and machine learning operations.

AI engineer adds one more layer of confusion. Sometimes it is just updated branding. Sometimes it really means product engineers building on top of large language models, vector databases, retrieval pipelines, prompt workflows, and evaluation systems. The title is growing for a reason. LinkedIn's 2025 AI labor market update (https://economicgraph.linkedin.com/content/dam/me/economicgraph/en-us/PDF/ai-labor-market-update-header-sept-2025.pdf) reported AI engineering roles approaching 7% of technical job postings on LinkedIn. But broad AI tool use is not the same as deep AI engineering capability. The Stack Overflow Developer Survey 2025 AI section (https://survey.stackoverflow.co/2025/ai/), JetBrains' 2024 State of the Developer Ecosystem (https://www.jetbrains.com/lp/devecosystem-2024/), and GitHub's 2024 enterprise developer survey on AI coding tools (https://github.blog/news-insights/research/survey-ai-wav

e-grows/) all point in the same direction: AI-assisted development is widespread, so its presence on a résumé is low-signal unless tied to concrete systems or outcomes.

Useful distinctions in specialist roles

ROLE	PRIMARY FOCUS	BEST SCREENING CLUE
Data analyst	Reporting and decisions	Dashboards, metrics, stakeholder analysis
Data engineer	Pipelines and data reliability	ETL/ELT, warehouses, orchestration
Data scientist	Statistical and predictive work	Models, experiments, methods used
ML engineer	Productionizing models	Serving, monitoring, machine learning operations
AI engineer	AI-powered product features	Large language model apps, retrieval, evals, integration
Security engineer	Risk reduction in systems	Which security domain they actually own
Product manager	What to build and why	Problem framing, prioritization, outcomes

For security roles, separate application security, cloud or infrastructure security, and governance-heavy work before you source.

Application security, often shortened to AppSec, is embedded in the software development lifecycle. The NIST Secure Software Development Framework (https://csrc.nist.gov/Projects/ssdf) and OWASP Top 10 guidance on insecure design (https://owasp.org/Top10/2021/A04_2021-Insecure_Design/) both support the same idea: secure software is not just scanner output. It includes secure design, secure coding practices, testing, and catching issues early. That is why AppSec candidates often come from software engineering backgrounds and work closely with developers.

Cloud or infrastructure security is different. Here the work is more about identity and access controls, cloud configuration, containers, networking, secrets management, logging, and incident response in platform environments. Governance-heavy security is different again: policy, audit, risk, compliance, control frameworks, and reviews. All are legitimate specialties. Bundling them into one role is how you end up interviewing everyone and liking no one.

Product managers belong in this discussion because they are often confused with business analysts or technical program managers. A product manager owns the problem, prioritization, and outcome. They decide what should be built and why. A business analyst is usually narrower and more requirements-focused. A technical

program manager is usually coordinating delivery across teams. Product managers may be technical, especially in platform or data-heavy areas, but their core value is not that they can talk to engineers without blinking. It is that they can connect customer need, business value, and trade-offs into a clear decision.

Questions I use to de-confuse specialist reqs

✓ What output does this role own: dashboard, pipeline, model, control, or roadmap?

✓ Is the work exploratory, operational, or production-critical?

✓ What tools are core versus incidental?

✓ What adjacent title keeps getting confused with this one?

✓ What would make a strong person from the wrong family still wrong here?

How to use role cards in real recruiting work

Role cards are only useful if they improve live recruiting. I use them across four moments: intake, sourcing, screening, and calibration.

At minimum, each card needs four fields: title, what they do, key skills, and common stacks. You can add a clarification field when the title is especially noisy.

Role card template

TITLE

The market-facing role name, plus close variants.

WHAT THEY DO

The core work and where it sits in the delivery process.

KEY SKILLS

- Core technical strengths
- Level signals
- Adjacent skills that may transfer

COMMON STACKS

Tools and platforms often seen, without treating them as the whole role.

In intake, the card stops you from accepting a title as a definition. Ask what this person builds, who uses their work, what breaks if the role stays open, and what is genuinely non-negotiable in the first months. One good intake question saves a great many bad screens.

A useful pattern is to separate fundamentals from environment. Fundamentals are the transferable core: building APIs, designing data models, writing test automation, improving reliability,

securing software. Environment is the local packaging: your cloud, frameworks, observability stack, and the homegrown scripts everyone swears are temporary. Hire only for environment and you miss strong people. Ignore it completely and you create ramp-up risk.

That same split helps in sourcing. Translate manager language into market language. "Improve deployment reliability" may point to SRE, platform engineering, or infrastructure-focused DevOps. If the so-called DevOps role is really internal platform work, say that on the card. The market is getting more specific here, and the 2024 DORA / State of DevOps report (https://dora.dev/report/2024) plus CNCF's 2025 platform engineering certification launch (https://www.cncf.io/blog/2025/06/15/introducing-the-certified-cloud-native-platform-engineering-associate-cnpa-community-driven-certification-for-platform-engineers/) both reflect that.

In screening, use the card to test for the work rather than résumé cosmetics. A candidate with generic titles such as "software engineer" throughout may still be a strong mobile engineer, backend engineer, or test automation specialist. Ask what they built, what systems they touched, what problems they solved, and what they owned end to end.

This is also where acceptable adjacencies matter. A backend engineer moving into data engineering may be viable if they have real pipeline exposure and strong SQL depth. A manual tester may be viable for an SDET role if they have been writing serious

automation. A frontend engineer with strong TypeScript and product sense may be a better full-stack fit than a nominal full-stack candidate whose backend work was light. A useful role card should show likely transfers, not just the ideal pedigree.

AI belongs on some role cards, but usually as context rather than identity. If the job truly includes retrieval systems, model evaluation, or machine learning infrastructure, put that on the card. If the team merely uses coding assistants, do not let that distort the search.

Calibration is where role cards earn their keep. After the first few screens, update the card with evidence. Which requirements were real? Which nice-to-haves were fantasy shopping? Which titles brought the right people, and which brought noise? I have seen searches improve quickly once the team stopped asking for a vague title and started describing the actual shape of the work.

Keep role cards current

- ✓ Review after intake and again after 3 to 5 screens
- ✓ Separate core skills from local stack requirements
- ✓ Add acceptable adjacent backgrounds
- ✓ Note title variants that are noisy or misleading
- ✓ Update AI expectations only when responsibilities changed

Keep the process light. A useful one-page card updated regularly is far better than a grand taxonomy nobody opens.

Conclusion

If there is one idea I want you to keep from this book, it is this: titles are hints, not definitions.

Once you start translating roles into actual work, technical hiring gets much easier to reason about. You stop overreacting to stack keywords, you ask better intake questions, and you reject fewer good people for title-related nonsense. That does not remove ambiguity completely. This is tech hiring. Ambiguity will always find a way in. But it becomes manageable.

In my experience, recruiters do not need a perfect taxonomy. We need a working one. A short role card, a few sharp clarifying questions, and the discipline to separate ownership from buzzwords will get you much further than another expensive tool with a confidence score.

If you are reading through the rest of this series, think of this book as the reference layer underneath the others. The better you get at mapping titles to work, the stronger your sourcing, screening, and hiring conversations become everywhere else.

Glossary

- **AI (Artificial Intelligence)** — Software techniques that let systems generate, predict, classify, or reason from data. In hiring, the term is often used too broadly.
- **AI engineer** — An engineer building AI-powered product features, often using large language models, retrieval, and evaluation systems.
- **API (Application Programming Interface)** — A defined way for one piece of software to communicate with another.
- **AppSec (Application Security)** — Security work focused on making software itself safer through design review, secure coding, testing, and early risk detection.
- **AWS (Amazon Web Services)** — A major cloud platform used to run applications, store data, and manage infrastructure.
- **Backend engineer** — An engineer focused on server-side logic, APIs, data, integrations, and the systems behind product features.
- **Browser automation** — Tools and scripts that automatically test web applications by interacting with them like a user would.
- **CI/CD (Continuous Integration / Continuous Delivery)** — Automated processes that help teams test, integrate, and ship code more reliably.

- **Cloud engineering** — Engineering work focused on cloud infrastructure, environments, networking, account setup, and related systems.
- **Component design** — Building reusable user interface pieces in frontend applications.
- **Containers** — Lightweight packages that bundle an application and what it needs to run consistently across environments.
- **CSS (Cascading Style Sheets)** — The language used to control layout and visual styling in web applications.
- **Data analyst** — A specialist who turns data into dashboards, reports, and answers for business decisions.
- **Data engineer** — A specialist who builds and maintains data pipelines, warehouses, and data quality systems.
- **Data scientist** — A specialist focused on statistical analysis, experiments, predictive modeling, or other advanced analytical work.
- **Database** — A system for storing and organizing data.
- **DevOps** — Originally a way of working that joins development and operations; often used as a job title for infrastructure and delivery-focused engineering.
- **End to end** — Work that covers a full feature or process from one side of the system to the other.
- **ETL/ELT (Extract Transform Load / Extract Load Transform)** — Common ways of moving and reshaping data between systems.

- **Flutter** — A framework for building mobile applications across platforms from one codebase.
- **Frontend engineer** — An engineer focused on the browser-based product experience, including interface behavior, state, and performance.
- **Full-stack engineer** — An engineer who works across both frontend and backend, though usually with more depth on one side.
- **GitHub Copilot** — An AI coding assistant used by many developers during software development.
- **Infrastructure as code** — Managing infrastructure through version-controlled code rather than manual setup.
- **Integration test** — A test that checks whether multiple parts of a system work correctly together.
- **iOS** — Apple's mobile operating system.
- **JavaScript** — A widely used programming language, especially in web development.
- **Kubernetes** — A system for running and managing containers at scale.
- **Large language model** — A type of AI model trained on large amounts of text and often used in chat, generation, and search applications.
- **Machine learning operations** — The systems and practices used to deploy, monitor, version, and maintain machine learning models in production.

- **Metrics** — Quantitative measures used to evaluate performance, outcomes, or trends.
- **ML engineer (Machine Learning Engineer)** — An engineer who turns models into reliable production systems.
- **Mobile engineer** — An engineer focused on building applications for phones and tablets.
- **Observability** — The tools and practices used to understand what a system is doing in production through logs, metrics, and traces.
- **Objective-C** — A programming language still seen in some older iOS applications.
- **Orchestration** — Coordinating automated jobs or systems so they run in the right order and under the right conditions.
- **Platform engineering** — Building internal systems, tools, and infrastructure paths that make software teams easier and faster to operate.
- **PostgreSQL** — A popular relational database.
- **Product manager** — The person responsible for deciding what should be built and why, based on user need, business value, and trade-offs.
- **Python** — A programming language used widely in backend, data, automation, and machine learning work.
- **QA (Quality Assurance)** — Work focused on validating that software behaves as expected.
- **React Native** — A framework for building mobile apps using web-style development tools.

- **Regression testing** — Testing to make sure new changes did not break existing behavior.
- **Retrieval pipeline** — A system that finds and feeds relevant information into an AI application at runtime.
- **SDET (Software Development Engineer in Test)** — An engineer who builds and maintains automated testing systems.
- **SDK (Software Development Kit)** — A set of tools and libraries developers use to build for a platform or service.
- **Service Level Objective (SLO)** — A target for how reliable or responsive a production service should be.
- **Site Reliability Engineering (SRE)** — Engineering focused on keeping production systems reliable, available, and observable.
- **SQL (Structured Query Language)** — The language commonly used to query and manage data in databases.
- **State management** — How a frontend application keeps track of changing data and user interactions.
- **Swift** — A programming language commonly used for iOS development.
- **Terraform** — A popular infrastructure-as-code tool used to define and manage cloud resources.
- **TypeScript** — A typed version of JavaScript commonly used in modern web applications.
- **Vector database** — A database designed to store and search embeddings, often used in AI and retrieval systems.

- **Vue / Angular / React** — Common frontend frameworks used to build web applications.
- **Warehouse (data warehouse)** — A central system for storing and querying large amounts of structured data for analysis.

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 — 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.