karat^

How to create a structured scoring rubric for technical interviews

Structured scoring rubrics make it possible for hiring teams to evaluate candidates consistently.

They put candidates who meet with different interviewers on a level playing field by standardizing what competencies are being evaluated, as well as providing consistency in language and rating scales. They help interviewers and hiring managers by making it clear which competencies matter. This helps guide which questions to ask in the interview, and where to spend time evaluating the candidate.

Each scoring rubric will be unique, depending on the job's roles and responsibilities, but here is a general kit to get you started:

Steps for building a rubric that interviewers can easily fill out during the interview:

1. Identify what competencies are both relevant and important to assess during the interview
2. For each competency, list observable behavior and results as checkboxes (select all) and/or radio buttons (choose one)
3. Write down an "algorithm" to help interviewers summarize a completed rubric into a single conclusion

For example, if technical communication is a relevant competency you might list "Technical Communication" on the rubric with a specific scale:

Technical communication

- ◉ Notably high quality communication
- ○ Clear explanations
- ○ Confusing or disorganized explanations
- ○ Notably negative communication

Even better is to match specific behavior to these assessments. Suppose you ask a candidate to explain their approach to solving a programming question:

Technical communication

- ◉ Approach included all relevant data structures and algorithm components, and was clear, easy-to-follow (i.e. step-by-step) and succinct
- ○ Approach made sense
- ○ Approach was hard to follow, disorganized and/or vague, and required follow-up questions to piece together important points
- ○ Poor interaction, i.e. candidate responded poorly to follow-up questions or was condescending, rude, or uncommunicative

What if instead of solving programming problems, you ask informational questions that explore how broadly or deeply the candidate knows a specific language or area of expertise? Also, what if you wanted to provide checkboxes for interviewers to record multiple observations, while also making it clear how that impacted the overall assessment? Here is an example of a competency, still shown as a "best to worst" scale, where interviewers can select multiple observations:

Degree of expertise

- ☐ Answers were on-topic and organized well, for example enumerated points and highlighted takeaways
- ▣ Answers were to the point and provided relevant details
- ▣ Candidate responded to follow-up questions with interesting insights
- ☐ Answers were fine
- ☐ Answers were hard to follow, disorganized
- ☐ Answers were vague and required numerous follow-up questions to piece together important points
- ☐ Poor interaction; i.e. candidate responded poorly to follow-up questions or was condescending, rude or uncommunicative

You could also mix and match. Suppose we wanted to assess how much of an expert a candidate is by asking them behavioral questions about past work. Here we have defined five levels of expertise, followed by some additional checkboxes to indicate notably positive behavior regardless of level.

Degree of expertise

- ○ Defines technical practices and standards across the engineering organization
- ◉ Makes major technical decisions, or designs new systems or major components
- ○ General expert who can fix any problem within their area of responsibility
- ○ Comfortable with the technology and/or expert in one significant domain
- ○ Coming up to speed

- ▣ Candidate evangelizes for improvements in technical best practices
- ▣ Candidate frequently helps other engineers complete their development tasks
- ☐ Candidate discusses a case in which their investigation into new technology meaningfully impacted their company's business (not just engineering)

By describing what behaviors are meaningful to the assessment, interviewers are explicitly guided towards meaningful indicators. If you notice interviewers coming to debriefs with negative opinions based on irrelevant, noisy indicators, you might explicitly guide interviewers to not consider those behaviors:

Candidate behavior during an interview that has no bearing on job competence or confidence, and thus is considered noise

- Presence or absence of upspeak (this is the raising of one's intonation at the end of sentences; it's common in American and Australian English speakers, especially young people and women, and bears no relationship to content, competence or confidence)
- Shyness (if specific leadership skills are relevant and important, they will be explicitly noted in the question guide and rubric)
- Use of filler words like "um" and "like", small amounts of rambling / blanking, and other behavior that may simply be the result of nervousness and stress
- Presence or absence of requests for validation or clarification, especially when specific guidance has not been made crystal clear i.e. informing the candidate that the next question they are asked will be intentionally vague or misleading in order to evaluate how they handle ambiguity

This is to reduce the noise that results from our pervasive historical culture of both:

a. Encouraging women and people of color to accommodate others, often by penalizing assertiveness as “aggressive,” “angry”, or “crazy”

b. Devaluing so-called feminine and so-called non-white behavior, which is where biases around “the glass ceiling” phenomenon come from

So far we have addressed only communication. Another relevant and important competency you’ll likely want to assess is the quality of the answers. Separating competencies into distinct items on the rubric allows you to make a more intentional hiring decision; for example:

- Are you looking for a strong technical expert who can successfully share their ideas with other leaders, and effectively empower contributions from the rest of the team?
- Or are you looking for a deep expert with good enough communication?
- Or someone on the cusp of leveling up who can fit new information into clean mental models and who works well with mentors?

A rubric that is built around explicit competencies, and does not conflate communication abilities with technical skills, will decrease both your false positives and hidden false negatives.

Here is an example of how you might structure a rubric to assess the quality of an implementation separately to its correctness.

Implementation quality

- ○ Notably well organized code, i.e. problem broken down into single purpose methods and logical step-by-step flow that is easy to debug and extend
- ◉ Code is fine for the candidate to understand and get a working solution relatively straightforwardly
- ○ Code confused candidate during implementation or debugging, which led to a helpful refactor
- ○ Code would be difficult for new teammates to grok and maintain, but candidate could not identify helpful refactors when asked
- ○ Code contains syntax and compilation errors

Implementation correctness

- ○ **Fully working solution**
- ○ **Close but time:** All major components of a working solution have been implemented. Candidate has correctly explained next steps, which you believe wouldn't have bugs. Select this option when you believe the candidate in on the cusp but unfortunately ran out of time. Note: If you and the candidate can go over time by five minutes to validate that the candidate would get to a fully working solution within five minutes, please do; if not, still select this option if it fits best.
- ◉ **Close but stuck:** Candidate has a solution that works on some input, but not others, and you believe it would take more than five minutes to identify the problem and implement a working solution. Select this option when a candidate has spent some time trying to debug an error and runs out of time while still stuck.
- ○ **Significant partial:** Candidate has not implemented one or more key components of the solution, i.e. a solution that does not successfully run on any input, or is missing a key data structure or data flow to run on some input, or is on the right track but contains major flaws. Conceptually the implementation contains big unknowns (~50%) in demonstrating a successful implementation.
- ○ **Trivial partial:** Candidate wrote some code, but did not make meaningful progress towards a working solution.

In coding interviews, if interviewers can provide hints when candidates are stuck, don't forget to add checkboxes to capture what hints were utilized. You can also add checkboxes for giving clarification and encouragement in order to make it explicit how certain behaviors do—or don't—impact the assessment conclusion.