# Technical Interview English

English answer patterns for technical interviews — the STAR method, thinking aloud, and system design phrases.

https://coderslingo.com/resources/cheatsheets/technical-interview/

---

## Clarifying questions (before you code)

*Asking good questions is a signal, not a weakness. Confirm scope before writing a line.*

- **Before I start, can I clarify a few things about the requirements?**
- **Can I assume the input is always valid, or should I handle malformed input?**
- **What's the expected size of the input — should I optimise for very large [n]?**
- **Are there any constraints on memory or time complexity?**
- **Should I prioritise readability or raw performance for this one?**
- **Just to confirm my understanding: the input is [X] and the output should be [Y]?**

## Thinking aloud while coding

*Interviewers grade your reasoning, not just the final answer. Narrate your approach.*

- **Let me start by restating the problem in my own words.**
- **My first instinct is a brute-force solution — let me get something working, then optimise.**
- **I'm considering two approaches: a hash map for O(1) lookups, or sorting first.**
- **The trade-off here is time versus space — the hash map is faster but uses more memory.**
- **Let me handle the edge case where the array is empty.**
  *Naming edge cases out loud is a strong positive signal.*
- **I'll come back to error handling once the happy path works.**
- **Let me trace through a small example to check my logic.**
- **I think this is O(n log n) because of the sort — let me confirm.**

## Recovering from a mistake

*Everyone makes mistakes under pressure. How you recover is what's actually being tested.*

- **Let me reconsider — I don't think that handles [case] correctly.**
  *Catching your own bug is a strength, not a weakness.*
- **Actually, there's a cleaner way to do this. Let me refactor.**
- **I made an off-by-one error there — let me fix the boundary.**
- **Good catch. Let me walk back to where the logic diverges.**
- **I'm a bit stuck — can I talk through where I think the issue is?**

## STAR method for behavioural answers

*Structure every behavioural answer as Situation, Task, Action, Result. It keeps you concise and impactful.*

- **Situation: We had [context — a failing deploy, a tight deadline, a flaky service].**
- **Task: My responsibility was to [specific goal you owned].**
- **Action: So I [the concrete steps you personally took — use 'I', not 'we'].**
- **Result: As a result, [quantified outcome — and what you learned].**
  *Always finish with a measurable result and a takeaway.*

## Behavioural question openers

*Common prompts and a confident way to begin each.*

- **"Tell me about a time you faced a difficult bug…" !’ "Sure — last year we had an intermittent**

**production issue where…"**

- **"Tell me about a conflict with a teammate…" !’ "I had a disagreement about [technical choice]. Here's how we resolved it…"**
- **"Describe your biggest technical challenge…" !’ "The hardest problem I've owned was [X]. The challenge was…"**
- **"Tell me about a time you failed…" !’ "Early in a project I [mistake]. What I learned was…"**
- **"Why do you want to work here?" !’ "Three things drew me in: [product], [tech], [team / mission]…"**

## System design phrases

*Drive the conversation: requirements !’ high-level design !’ deep dives !’ trade-offs.*

- **Let's start with the requirements — functional and non-functional.**
- **What scale are we designing for — requests per second, data volume, users?**
- **Let me sketch the high-level architecture first, then we can go deeper.**
- **The main components are [clients], [API layer], [service], [data store], [cache].**
- **The trade-off here is consistency versus availability — given the use case, I'd lean [eventual / strong].**
- **To scale reads, I'd add a cache and read replicas; for writes, I'd shard on [key].**
- **The bottleneck will likely be [X]; here's how I'd mitigate it.**
- **Let me call out the failure modes and how the system degrades gracefully.**

## Sample STAR answer

```
Q: "Tell me about a time you improved system reliability."

Situation: Our checkout service was throwing 5xx errors during
peak traffic, roughly twice a week.

Task: I was asked to find the root cause and stop the incidents
before the holiday sale.

Action: I added tracing, found a connection-pool exhaustion under
load, introduced a circuit breaker, and tuned the pool size with
a load test to validate the fix.

Result: Error rate during peak dropped from ~2% to under 0.1%,
and we had zero incidents through the sale. I also wrote a runbook
so on-call could spot the pattern early next time.
```