CODERPAD · TECHNICAL HIRING GUIDE · 2026

System Design Interviews. A guide for fair, skills-based hiring.

AI has made coding knowledge a commodity. The engineers who matter can do what AI can't: design the systems that AI runs on. This guide covers how to run system design interviews that surface real engineering judgment — and why the format matters more than ever.

WHY SYSTEM DESIGN MATTERS MORE THAN EVER

Generative AI hasn't removed the need for rigorous evaluation — it's shifted where the signal lives. Coding questions are easier to complete with AI assistance. System design interviews are now the most defensible signal of true engineering judgment, because they test what AI cannot do: reason through ambiguity, make context-sensitive trade-offs, communicate decisions to stakeholders, and architect systems that survive contact with reality.

What's changed

AI can produce correct-looking code quickly. Senior engineers must now evaluate, integrate and adapt that output. Hiring now depends on judgment, not just implementation speed.

What hasn't changed

Strong design still requires experience and real trade-offs. Real systems involve competing constraints. Clear thinking and communication still separate strong candidates.

WHAT SYSTEM DESIGN INTERVIEWS ACTUALLY ASSESS

The goal is not to see if a candidate can produce the "right" design — there is rarely one. What you're evaluating is **how they think** and whether that thinking holds up under pressure, constraints, and iteration.

DIMENSION	WHAT YOU'RE LOOKING FOR	RED FLAGS
Problem Framing	Clarifies scope before jumping in; asks about scale, users, constraints	Immediately starts designing without asking questions
Systems Thinking	Identifies components, trade-offs, failure modes, and how pieces interact	Treats the problem as isolated; no discussion of failure or load
Communication	Narrates decisions clearly; explains why, not just what	Silent design with no reasoning shared
Adaptability	Responds gracefully when constraints change or interviewer redirects	Digs in on original design despite new information
Depth on Demand	Can go deep on any component when pressed	Shallow on everything; surface-level across the board

Unlike coding interviews where the problem is well-defined and the answer is verifiable — system design interviews are deliberately open-ended. The interviewer isn't looking for the right answer. They're watching how a candidate navigates ambiguity, makes trade-offs, and communicates under pressure. The structure below is what a well-run **45–60 minute session** looks like.

PHASE	TIME	WHAT'S HAPPENING	WHAT YOU'RE EVALUATING
1. Clarify	5–10 min	Candidate asks questions before touching the board. What's the scale? Who are the users? What does success look like? Read-heavy or write-heavy?	Do they know what to ask before they start building? Jumping straight to design is a red flag.
2. Estimate	5 min	Back-of-envelope math. How many requests per second? How much storage? This is about reasoning under uncertainty before committing to an architecture.	Can they size a problem? Do they understand that 1M users and 100M users are fundamentally different design challenges?
3. High-level Design	10–15 min	Draw the boxes. Client, load balancer, servers, database, cache. What talks to what. Rough and fast is fine — the skeleton matters, not the polish.	Does the shape make sense? Are critical components present? Are there obvious gaps in how data flows?
4. Deep Dive	10–15 min	Interviewer picks a component: 'Walk me through how your database handles 10k concurrent writes.' Candidate zooms in without losing the whole picture.	Can they go deep on demand? Do they understand failure modes, bottlenecks, and the cost of their design choices?
5. Trade-offs	5–10 min	Candidate explains what their design sacrifices. What would they do with more time? Where are the weak points? What would change if scale 10x'd overnight?	Self-awareness and engineering maturity. The best candidates can critique their own design as clearly as they built it.

WHITEBOARD VS. CODERPAD: KEY DIFFERENCES

	WHITEBOARD / UNSTRUCTURED	CODERPAD
Consistency	Achievable with discipline and well-trained interviewers, but standardized prompts are difficult to track and hold interviewers accountable to	Built into the platform. Easy to compare multiple candidates on the same prompt and align interviewers around real examples, not interpretations
Diagramming	Faster and more natural for most candidates — boxes, arrows, and edits happen fluidly without learning a new tool	Diagrams sit alongside code, notes and discussion. Candidates can mix visuals with lightweight pseudocode or APIs, reflecting how systems are actually designed — collaboratively
Documentation	Ephemeral by default. Photos help, but whiteboard artifacts are hard to share or revisit in debrief	Preserved automatically. The full design history is reviewable by anyone in the loop before the debrief even starts
Bias Risk	Harder to control. Scoring relies on interviewer notes and memory unless a structured rubric is actively enforced	Structured scoring tied to observable, shareable artifacts makes calibration easier across interviewers and hiring cycles
Candidate Experience	Familiar and low-friction for most engineers. Can feel more like collaborative problem-solving	More representative of remote/async work. Levels the playing field for candidates who interview poorly in-person

The quality of the prompt is as important as the quality of the evaluation. A bad prompt either leads every candidate to the same solution (no signal) or is so open it's impossible to evaluate consistently.

What makes a strong system design prompt

It has a clear user need, meaningful scale constraints, and at least one non-obvious trade-off baked in. There should be no single correct answer, but there should be several clearly wrong ones. The best prompts are grounded in real problems your team has actually faced.

STRONG PROMPTS TEND TO INCLUDE:

- A user-facing outcome, not a technical spec ("users should be able to search their messages in under 200ms")
- Scale parameters that matter ("starting with 1M users, expecting 50x in 18 months")
- At least one constraint that forces a real trade-off ("you have a 3-month runway to ship this")
- Room to go deep on multiple components (caching, storage, API design, reliability)

CLASSIC PROMPTS AND WHAT THEY TEST:

- **URL Shortener (Bit.ly) — caching and hashing, good starter**
- **Messaging System (WhatsApp) — real-time delivery, fan-out, read receipts**
- **News Feed (Twitter/Instagram) — write vs. read optimization, ranking**
- **Rate Limiter — deceptively simple, gets deep fast**
- **File Storage (Dropbox) — chunking, deduplication, sync conflicts**

Prompt libraries eliminate the consistency problem

A great prompt that lives in one interviewer's head isn't standardized. CoderPad lets teams build a shared prompt library where the question, constraints, and rubric are delivered identically to every candidate. The prompt doesn't drift. The bar doesn't shift between interviewers or hiring cycles.

Sessions turn prompts into data

When you run the same prompt across 20 candidates in CoderPad, you accumulate a corpus. You can see whether the prompt is too easy (everyone converges on the same design), too hard (nobody gets past high-level), or subtly biased in a direction you didn't intend. That kind of signal is invisible when interviews live on whiteboards and scattered notes.

WORKED EXAMPLE — "DESIGN A URL SHORTENER" WHITEBOARD VS. CODERPAD

The same 45-minute interview, run two ways. The prompt: design a URL shortening service like Bit.ly that handles **100M active links** and **10B redirects per month**.

	AT A WHITEBOARD	ON CODERPAD
Clarify	Verbal conversation. No record of what was asked or agreed — it lives in two people's heads.	CoderPad's transcription captures it automatically. Agreed constraints are on record without anyone needing to stop and write them down.
Estimate	Easy to lose the thread — candidates often skip this step because there's no natural place to capture it.	The transcript preserves the reasoning, so reviewers can see whether the architecture matched the stated scale.
High-level Design	Fast and fluid. Freehand diagrams are where whiteboarding genuinely shines. No UI friction at all.	Diagrams are persistent and shareable. Candidates unfamiliar with the tool may slow down, but the gap narrows with familiarity.
Deep Dive	Good interviewers take notes; distracted ones lose the thread. No easy way to annotate the diagram mid-conversation.	Interviewers can annotate the shared canvas in real time. Pseudocode lives next to the diagram — not described verbally and then lost.
After the Interview	Someone photographs the whiteboard. Quality varies. Hiring committee relies on whatever each interviewer remembers.	Full session — transcript, diagram, written notes — is available to every interviewer before the debrief. Pattern analysis across candidates becomes possible.

Give candidates a 2-minute orientation to the canvas before starting the clock. The benefits of a persistent, shareable session far outweigh the brief onboarding.

The most forward-thinking engineering teams are evolving system design interviews into a two-act assessment that doesn't stop at the diagram. After a candidate designs the system, ask them to build the most critical component — right there, in the same session. Not as a gotcha. As a signal. Because here's what the whiteboard alone can't tell you: **was the design real, or was it theoretical?**

WHY THIS CHANGES WHAT YOU'RE ASSESSING

AI tools can generate plausible-looking code quickly. The question is no longer 'can they write it' — it's whether they can direct AI purposefully, evaluate what it produces, catch its mistakes, and make sound judgment calls when the output drifts from the design intent. That's almost exactly what senior engineers do every day in 2026.

The Architect

Does their design hold up when they try to implement it? Theoretical designs collapse fast when code has to actually run.

The AI Director

Can they prompt effectively, evaluate output critically, and push back when AI goes in the wrong direction?

The Doer

Do they know what to build and what to skip? Scope judgment under time pressure is a senior engineering skill.

HOW IT RUNS IN PRACTICE

The design-to-build interview extends the standard 45-minute session by **20–30 minutes**, or runs as a dedicated second stage. The handoff moment is the key transition:

STAGE	WHAT HAPPENS
Design (as normal)	Candidate clarifies, estimates, and produces a high-level architecture. Standard system design interview — nothing changes here.
The Handoff	Interviewer says: 'Pick the component you consider most critical or most interesting. You have 20 minutes and access to AI tools. Build it.' The candidate chooses what to implement — **that choice itself is a signal.**
Build with AI	Candidate uses AI coding tools to implement their chosen component. Interviewer watches how they prompt, what they accept, what they question, and whether the output aligns with the design they just described.
Review & Defend	Interviewer asks the candidate to walk through what was built, where AI helped, where it went wrong, and what they'd do differently with more time.

What this format exposes

Candidates who design confidently but can't implement anything. Candidates who implement quickly but build the wrong thing. And the rarest signal: candidates who design, build, and critically evaluate AI output — the engineers who will still be exceptional five years from now.

The right question to ask

Not 'can they build it without AI?' That's the wrong bar. The right question is: 'When they use AI to build, do they produce something good?' Judgment, taste, and architectural thinking still separate great engineers from average ones.

> *CoderPad is the only technical interviewing platform where both halves live together. The diagram the candidate drew is right there alongside the code they're writing. The transcript captures the full arc — from the first clarifying question through the final build review.*
>
> **CoderPad** — System Design + Build, same session

Ready to run better system design interviews?

Ask your CoderPad CSM about structured system design sessions, shared prompt libraries, and the design-to-build format. Everything you need is already in the platform.