SDE Phone Screening Guide

[Jacob Kjelstrup – November 14, 2004.]

This document is a short guide and set of "best practices" for doing Amazon SDE phone screens for Supply Chain. My goal is to provide some general suggestions and a sample "flow" rather than to present a cookbook of questions and answers.

Phone Screens Are Hard!

Yes, they are! It's a lot more difficult to interact with the candidate when you can't control the environment. You won't be able to provide a whiteboard or watch the candidate turn design into visible code. However, it's crucial to do good phone screens as it costs us money and wasted time to put candidates through on-site interview loops.

A decent phone screener will have a ratio of at least 20% offers per positive screen, so even with screening there's a lot of overhead in order to get some job offers sent out!

There is hope, though. You will definitely get better with time, but you need to practice periodically, so don't get scared away!

First Screen vs. Second Screen

So far, we've gone through a single, generic phone screen and analyzed it for best practices and suggestions. In reality, SCOS will generally have two technical phone screens before bring the candidate in. Should these phone screens be different at all? In a word, yes!

1. The first screen should focus more on core CS skills, coding, and passion. After you're done with the introductory stuff ("Why do you want to work for Amazon? What can Amazon do better?"), describe what we're looking for and then dive into these core skills. If they don't do well at all, close out gracefully (as explained previously), mark the candidate for rejection, and let HR handle it from there. The first screen can be the same for all SDE candidates from SDE-1 to Principal. Only schedule a second phone screen if you're inclined. Avoid the "I'm not really inclined, but maybe I should have Bob take a look" syndrome. We'll be screening a lot of candidates, so let's be efficient with our time.

2. The second phone screen should definitely focus on design, problem solving, and evidence of delivering stuff. Probe a project on the resume for proof of delivering. For SDE 1 and 2, make them design something involving design patterns or data structures. For SDE 3 and Principals, do something with more complexity, perhaps involving distributed systems. Try to work in some coding somewhere in the design problem. It's very useful to get two coding data points, even if they only have to code a simple function within their design solution. If the candidate passes this second phone screen, enter the proper annotation into MRT and move the candidate to the "schedule interview" step.

Minimum core coverage: There has to be some coding, preferably in both screens. There has to be some basic Object Oriented questions (see the "Design" section). And there has to be some probing into bits and bytes. I also recommend testing if they know basic scripting and why it will save time on things like looking for patterns in files.

Pre-Screen Prep Work

1. Always scan the resume before you make the call. You don't necessarily need to read it in detail. If you're going to ask the candidate about a particular project, this is a good time to figure out which project to dig into. I try to find a project that deals with a technology that I have some experience with. That will come in handy later! I also like to note which technologies the candidate lists on the resume. If a candidate claims to know something, it's fair game to go after that area.

2. Reading the resume also constitutes a kind of sanity check that the candidate matches the job. If you're interviewing for an SDE and the candidate has been VP of Engineering with 400 people reporting to him for the past 10 years, something's wrong. Something like that actually happened to me once, so don't take it totally for granted.

3. Now that I know a little bit about the candidate, I prepare my questions. I like to do this before I actually make the call so that the actual interview goes smoother.

4. Get a notebook and pen ready. I write down everything the candidate does (well, not every word, but you know what I mean), so that I can review it at any time.

Placing the Call

1. Find a reasonably quiet location. Our hotelling stations are good. Be on time. Remember that you're representing and selling Amazon in addition to screening the candidate. Make a positive impression!

2. I usually start by introducing myself briefly and asking them if they were expecting my call. If someone else is with you and will be co-interviewing, introduce them at this time.

3. Verify that the job you're screening them for is what they're expecting. Tell them that you're going to screen them for a software development position. Some people like to tell the candidate how long the interview will last. Personally, I don't do this since I want the freedom to terminate the interview early if things are going poorly. (And even if it went poorly, remember that you still want them to feel good about the process.)

Part 1: Soft Stuff

1. It seems a bit harsh to jump directly to: "Hey, tell me why you might declare a C++ destructor virtual?" Instead, you might want to ask them why they want to work at Amazon. If they answer with "I don't care, I just need a job" or "I got laid off", that's the wrong answer. I'm looking for some passion, and frankly for some smarts. There are lots of right answers to this question! If the candidate doesn't have the energy to answer this question properly and just blows you off, that's not the kind of coworker you want.

2. Other good introductory questions are: "What do you like about Amazon?" or "Do you have any suggestions for how we can make our website better?" Remember, you're looking for passion and energy more than a hardcore analysis. Be quick about it, though. You should only take a minute or two for this.

3. One Amazon Bar Raisers likes to ask: "What originally got you interested in computer science". Amazingly, it can help weed out some losers: "high pay", "I'm not actually sure that I like CS", "lots of jobs", "I didn't know what else to do", etc. You'd be surprised at what you might find out. Here, you're looking for warning signs that the person is not committed or is not a real builder. You can them explore these warning signs further later in the interview.

4. Next, you might ask them about their current project. Or, if you selected a particular project in your resume prep, this is the right time to head in that direction. Your goal here is to figure out if they actually understand their own project and if they made any kind of meaningful contribution. Since you picked a project that you have some familiarity with yourself, you should be able to ask some very directed questions. Don't let the candidate answer only at a high level

(although the candidate should be able to give a succinct high-level overview to start). Again, your goal is to see if they understand the details, so you'll need to quickly dig deep. The "soft stuff" is only one part of the interview, so don't get bogged down. If you're stuck, extract yourself and go on to phase 2.

Phase 2: Design Skills

At this point, the candidate is warmed up and should be able to do some serious work. I now give them a design question to chew on. One of my favorites (assuming they have databases or SQL on their resume) is to dig into a schema design question. I'll give them a use case like: "Pretend you're hired by a startup dot-com company to design a blah-blah-blah system…" What I want from them at this point is to design a couple of database tables and put some reasonable columns into them. I then give them a particular use case: "Write a query to find the zip codes for all the orders within the last 24 hours." I'll be checking to see if they can join two tables (customers and orders in this case) and write a basic query. I'll also ask them what they'd do if the query runs slow in order to see if they understand anything about query optimization. If they mention things like "indexes", then immediately probe and see if they know what an index really is and what it does. Make sure they don't throw buzzwords around.

Another good design question might center on how to build a reliable messaging system. Here you'd be looking for the candidate to understand the core issues of the problem (how to detect errors, how to re-drive messages, how to make it efficient, etc). Again, once a basic design has been proposed, probe deep into a couple of things rather than stay at the higher level. That's important since you're trying to ascertain whether the candidate can actually build something and not just talk about it. I've interviewed lots of candidates who seem very promising at the high level, but I can't get anywhere with them when we get into the details.

There is an unlimited number of good design questions you could ask. You could probe into tree structures and see if they know the how/why of them. Ask them to design something that requires more than textbook knowledge, though.

Or you could see if they understand a concept like deadlocks and whether they can design a deadlock prevention system for a use case that you'll give them. You get the idea.

I suggest that you prepare two or three of these "design-then-go-to-details" questions ahead of time. At prep time, select one that should match the candidate's background. You want to make sure that they know what they claim to know at this stage, not present them with radically new problems to solve.

One of the two phone screens needs to have some OO design questions. Make sure that they understand some of the basic concepts, like is-a vs. has-a (with examples), overloading vs. overriding, class vs. object, method vs. function, virtual vs. pure virtual, base class vs. derived class, inheritance, encapsulation, delegation, aggregation, and so on. Then, make them design something simple. What you're looking for in a screen is not an in-depth OO problem (like the elevator design question), but rather to weed out insane designs like "I'll have Person inherit from Arm, Leg, Head, and Body" and "I'll have Car and Motorcycle inherit from Garage". You're just looking to see if the OO lights are on.

Allocate no more than 20 minutes for this phase. You'll need to drill deep enough to satisfy your goals, but this is not the end of the interview.

Phase 3: Go Wide

The candidate might be a little tired at this point. Therefore, I now like to ask a bunch of quick and simple questions. These should aim at verifying basic knowledge and skills without going deep. Here are some C/C++ samples:

- What's the difference between pointers and references in C++?
- What's a static variable?
- What's the heap?
- What's the difference between call-by-reference and call-by-value?
- Do they understand the scope of variables and how the stack is used?
- What is polymorphism?
- Why is polymorphism better than simple collection of source files with subroutines and data structures?
- What's a virtual function and how do they work?
- What's the big-O for growing a vector?

You could also ask about UNIX command line tools. Or you could ask about regular expressions. Or threading. Or garbage collection. Or synchronization primitives. And so on.

If an SDE claims to know UNIX, then go ahead and ask some kind of question like: "I have a log file structured like 'username<TAB>command' and I want to know how many times user Bob ran a command." Then let them solve it. If they want to write an OO C++ or Java program, they are not UNIX people. If they understand grep and regular expressions, that's a good sign.

Another "wide" topic that some experienced phone screeners will **REQUIRE** is bits and bytes. If candidates know nothing about this, they will likely not succeed. Here are some basics:

- What are the logical operations? (AND, OR, NOT, XOR.)

- What's the difference between logical-AND and bitwise-AND?
- What is 36 expressed in hexadecimal?
- Write an if-statement that tests whether the high order bit is set in a byte.

We also insist that a candidate has a basic understanding of the core data structures. They should have some knowledge of arrays, vectors, lists, trees, hashes, and graphs. They should be able to describe when they'd use a given data structure and give the reasons for why. They need to be able to express themselves in terms of big-O performance.

The important part here is to ask things that may seem a bit basic or high level, but it is stuff that the candidate should know. Once again, try to dig slightly deeper than the text book. Lots of candidates remember the buzz words, but don't understand the concepts. So, if you ask about polymorphism, make them give you examples of how they'd use their classes.

Phase 4: Coding

This phase is mandatory for SCOS! Always make the candidate code! Failure to write decent code is the most common source of onsite interview rejections. We should always do two technical phone screens, and both should include coding (in my opinion).

But how does one go about doing coding over the phone? Some people have implemented different solutions. There are two basic styles: make them code something up after the interview and mail it in, or make them do it in real time during the phone screen.

There is no right or wrong approach. All I'll say is that I personally prefer to do the coding questions in real time. It's a bit awkward, but I've always gotten it to work. This gives me the ability to question them interactively ("Why does the function return a pointer?"). I also get a better feel for how they approach the problem since they'll often ask clarifying questions or talk out loud about their ideas. It also prevents cheating (in case their hardcore roommate is standing by to help them code). I once actually heard a candidate start typing on the keyboard when I asked a coding question, but I don't think google'ing out some code in real time will help them much.

I start off by telling them that they should get paper and pencil ready because we're going to do some coding. Sometimes, they may not believe me (!), but I calmly explain that the two of us will be walking through the coding together.

I make them tell me everything, including the function's return code and arguments. It's critical to be clear about these kinds of details. They can't code a subroutine if they haven't declared a return type. As we go through the coding problem, I ask them for each and every line. Variable declarations are crucial, as are loop conditions, etc. All

these kinds of things help indicate whether they can actually construct some non-trivial code or not. Here, you want to make sure that they actually get the details right. Simply verbalizing the design is not good enough.

The coding questions should also require some basic structural design, but not too much given that this is a phone screen. I like coding questions like "reverse a C string" or "code itoa( )". Again, find your own set of coding questions and select one of them up front.

Make sure you've left enough time to finish the coding question. This is a crucial part of the screen.

Phase 5: Wrapping Up

Finally, it's time to wrap up! I usually thank them for their time and ask if they have any questions for me. After grilling them for an hour (that's how long my screens usually take), I figure that it is only fair for them to ask me some questions. Finally, let them know that their recruiter will be in touch with them shortly, usually within a few days.

If they have specific questions about when they'll be contacted or other concerns that are not immediately relevant to your phone screen, just refer them to the recruiter. That's easier than getting into HR's territory!

Calibration is Key!

Ok. So, you've asked all these questions and taken notes. How do you make the final decision whether the candidate is good or not? Sometimes it's quite obvious, of course. If they bomb big parts of the interview or they ace everything, your decision is obvious.

And since these are SDE interviews, they need to pass the coding question. No exceptions.

For the remaining cases (the majority!), calibration is the key. Hopefully, you've already asked similar questions in onsite interviews and have some idea about how the candidates should perform. Over time, you will build up a good understanding of where the bar goes for each of your core questions. I can't emphasize this enough! All of these questions do no good at all unless you have a clear understanding of what's a good or bad performance. Never go down any line of questioning unless you know what you're looking for and can thus grade the candidate.

Even though I've already said this, I'll say it again: It's super crucial to go somewhat deep at least once in each interviewing phase. If you don't, you won't be able to adequately calibrate the candidate. Lots of candidates can talk a good story about their database projects until the cows come home, but won't be able to explain to you how to optimize their queries. If lack of coding questions is the leading cause of phone screen failures, then staying entirely at the high level is the second worst offender. Don't fall into that trap!

In order to keep your calibration coefficient precise, you'll also need to monitor how the candidate does when the actual onsite loop has finished. You may find that you're too lenient or to hard in your calibration for certain questions. You may also find that you missed the mark completely on a question. This is how you'll tune your repertoire of questions and their associated calibration data. This is really important!

Things Not To Do

1. Don't let the candidate do most of the talking. One of the most common signs of a bad phone screen is when the candidate fills up the time by talking about what he/she knows and has worked on. No! Instead, drive the interview yourself and make the candidate answer the questions that you have prepared.

2. Never show off to the candidate. You're not there to show them how smart you are or how much you know. At best that's a waste of time. At worst, you'll turn away a good candidate. Therefore, ask the candidate questions that he/she should be able to answer. It's not your goal to blow away the candidate, but rather to probe the candidate's skills.

3. Don't torture the candidate. If they're hopelessly stuck, stop that line of questioning and go on to something else.

4. Don't finish any of your phases until you've gotten a clear indication of success or failure. This is closely related to the calibration issue.

5. Don't let the candidate confuse you with strange approaches and weird answers. As long as you ask good, basic design and conceptual questions, the candidate should be able to give a clear answer. If you can't understand the candidate's design even after repeated attempts at clarification, you'll need to deduct points for bad communications skills.

6. Don't make things up! For instance, if the candidate asks about visa status and work permits, it's probably better to refer them to HR than to take a guess. Just stay with the technical stuff.

7. Don't decline the candidate on the spot! If they're doing very poorly, cut the interview short, but let HR do the follow up. They're better at it then you are

most likely!

8. No clever questions. You're not out to test if they get some fancy trick that you picked up from one of your geeky friends. Those kinds of things are not good indicators of the candidate's abilities. It's almost impossible to calibrate the trick questions anyway, and that goes against the core principle of this tutorial.

Writing Feedback

It's important to write good feedback. If you feel that another phone screen is needed, say so. (And also alert the recruiters, of course.) Give suggestions about which areas the next screener should focus on. Even if there are no more screens to be done, good feedback is important for the onsite interviewers, who should read your notes and use them to avoid or probe various areas.

Ultimately, the purpose of this entire process is to make an inclined or not-inclined judgment. You're not done with the screen until you've made a choice. If you're near the end of the interview and you still have no clue about how you stand, you're in trouble. Try to ask a few detailed tie-breaker questions. You might also want to consider taking another 10 minutes if that's possible. If you're still not sure when everything is said and done, you're not inclined by default. Avoid the "I have some serious problems with the candidate, but she might be ok anyway so I'll vote inclined and see what happens next." I've seen many examples of this (even from myself!) and those things seldom work out in the end. Only vote inclined if you have a solid feeling that the candidate will succeed at Amazon.

When dealing with a tool like MRT, please remember to move the candidate on to the next stage (a second screen or onsite interview or rejection) and assign the task to our recruiter. If you don't do this, the candidate may sit around seemingly abandoned, which will not put him in a good mood. It also encourages other groups to poach.

The Bottom Line

The bottom line is to find excellent candidates for Amazon to fuel our growth. Therefore, don't focus just on rejecting candidates for the most minor faults. Our goal is not to reject candidates, but to hire the right ones! It's all about supporting our corporate (and local) goals through hiring!!!