Amazon Phone Screen Cheat Sheet

Author: **Steve Yegge**
Date: **April 8, 2003**
Version: **1.0**

What Is This?

This document is a quick list of things to remember before doing a phone screen, and it includes some sample questions at the end.

Everyone has his or her own phone-screening style. I'm not trying to mandate anything here. This is just my personal style and agenda for doing screens.

You should do lots of phone screens. If you feel like saying "I *hate* doing phone screens," just remember: everyone hates doing them. But they're important. And you get better at them with practice.

Learn how to screen resumes well, and it'll save you a lot of wasted time.

Before the Screen

1. Read the candidate's resume. You might only need to spend 1 minute on it.

2. Understand the position they're applying for, and the requirements for that position. Know why you're hiring the person.

3. Make sure you'll be available for the screen, and ask recruiting to reschedule if you can't make it. (Or better yet, find an alternate to stand in for you.)

4. Prepare a list of questions to ask.

Dos and Don'ts

Stuff to do:

1. Defer all questions you're not sure about to the recruiters. Tell the candidate "I don't know the answer to that, but I bet your recruiting contact at Amazon (the person who scheduled this phone screen) will know."

2. Make sure the candidate knows who to contact if they have further questions (usually the recruiter, but can be you)

3. Reserve 5 minutes at the end for letting them ask questions

4. Write up detailed feedback after the screen and send it to the recruiter

5. Always try to give the candidate a positive impression of the company, even if you're not interested in following up.

Stuff not to do:

1. Don't torture the candidate. If they don't know the answer to a question, make a note of it, and move on.

2. Don't type while the candidate is talking. It scares them. Take handwritten notes instead, and transcribe them later.

   Note: I break this rule all the time. I much prefer typing, since I can get better notes that way. However, I always ask the candidate for permission first, telling them that I write like a second-grader (which I do.) They're usually OK with it, and I try to type quietly during the screen.

3. NEVER end the screen until you've made a yes/no decision about whether to continue with the candidate (e.g. bring them in for interviews, or do another phone screen).

Making the Phone Call

Placing the call is usually easy, but there are some protocol points to remember.

1. Make sure you're in a quiet location.

2. Call the candidate at the scheduled time.

3. Introduce yourself, and ask to speak to the candidate.

4. Ask if the candidate was expecting the call, and if they have time for half an hour to an hour (depending on the candidate's experience level.) If not, reschedule the screen.

5. Make sure you and the candidate agree on what job you're screening for.

6. If someone is listening in on the screen, let the candidate know.

Types of Questions

There are several basic kinds of questions you can ask:

1. **Culture Fit**. You're trying to figure out if the candidate wants to work here, if they have common sense and common courtesy, if they have reasonable expectations about working hours, if they deal well with ambiguity, and so on. It's basically their "soft skills."

2. **Projects and Work Experience**. You're trying to see if candidates have accomplished difficult things, and if they can talk about them intelligently.

3. **Technical/Problem Solving**. This is where you find out whether the candidate has a sufficient range of experience and technical depth for the position.

Call Flow

You typically only have 30-45 minutes for a phone screen. You have two missions in life during the screen:

1. Give the candidate a positive experience.
2. Make sure you arrive at a yes/no decision by the end of the screen.

To accomplish these objectives, you should structure the call in separate sections that should flow together smoothly.

Warmup. Don't jump right into a bunch of hard technical questions. Spend two or three minutes asking the candidate easy stuff. Let them talk about their current context – ask what they've been working on lately, how they like it, fluff stuff. It's OK to ask job history stuff here, too, as long as you sound like you're asking purely out of interest.

Project Work. Since they just talked about what they're working on, or worked on recently, ask if they can tell you about one of the projects they mentioned. Pick one that sounds interesting, or if there isn't one, keep digging until you've found one that is.

During the project-work section, you should be asking detailed questions about the project architecture, the roles of the contributors, and the context for the project. It's a good idea to do an occasional "deep dive" and have them explain something particularly technical, and then come back up to the high level.

Technical. You should now be about 8 to 10 minutes into the screen, unless they had a really interesting project – interesting to you, not them. If the candidate is getting boring, or you're not getting the level of technical detail you need, or the candidate just won't shut up, cut them off. There's a trick to this. You start saying "uh-huh, uh-huh… yeah, yeah!" and wait for a pause, and then say (loudly): "GREAT! That's really cool. That sounds like it was a hard project. We can talk more about it later if we have time."

Now the candidate should be warmed up. Tell them you'd like to ask some technical questions, and that you'll cover a broad range of topics, just to get a feel for how much variety the candidate has had at school or work.

There are lots of sample questions at the end of this document.

Wrapup. After 20 to 25 minutes of technical questioning, you should have arrived at once of two conclusions. The set of possible conclusions consists of:

1. Yes.
2. No.

There aren't any other acceptable conclusions. Take any "maybe" and turn it into a "no". There are plenty of other candidates out there who can wow you, so don't compromise.

The best way to wrap up is to say: "OK! We've covered a lot of stuff – object-oriented design, some systems questions, some language questions, some project questions, some data structures stuff… I think that's pretty much everything I wanted to ask. I've got another five minutes left, though, so do you have any questions for me? Is there anything you'd like to know about Amazon, or what my group does, or anything like that?"

Works like a charm. Take notes on the questions they ask. Good candidates ask good questions. This is your time to sell the company. Doesn't matter if the candidate is good or bad – make sure you sound excited about the company and your role in it.

The hardest question to answer, and one of the most common, is: "What are the next steps?" That's a candidate's polite way of asking "Are you going to hire me, or am I going to go jump off a highway overpass into oncoming traffic?" It can be a fairly awkward moment.

Some people feel comfortable declining the candidate right there, on the spot. I'm not one of those people. It's acceptable to do it, if you're careful what you say. I'm not going to tell you what it is, though, since I don't know. Probably something like, "OK, well, while you've done a lot of interesting work, and you answered some of my questions correctly, I don't think there's an exact fit for your skillset at this time." Be prepared to answer when they ask "Why?" – that's also hard. But be honest.

I, however, prefer to let them down easy. Here's my standard blurb:

"Well, we've talked about a lot of stuff, and I've taken a lot of notes during our talk. I haven't had much time to think about them yet. I'm going to finish writing up the notes, and then talk it over with my team and with the recruiters. We'll decide within the next 2 days whether we're going to bring you in for interviews. If we do bring you in for interviews, it'll be an all-day thing: you'll have five or six 1-hour interviews with various people in the organization. If you come in for interviews, the recruiters will work with you to set up a date for it. Either way, we'll get back to you within 48 hours. If you don't hear from us, you should feel free to contact the recruiter and ask, since we occasionally lose things in the shuffle."

That usually mollifies candidates. You've told them that it'll be a group decision (which it should be – you can ping someone about the screen before sending the feedback to the recruiter), and that you'll get back with an answer quickly. Thank them for their time, and you're done!

Sample Questions

Culture Fit

Q: **Why do you want to work at Amazon?**

Good answers:

- I'm looking for challenging work, working with smart people
- I love your brand name. I've bought stuff from you and I am keenly interested in seeing how it works "under the covers".
- I have several friends who work here and they've convinced me that this is a great place to work.
- I have an e-commerce background and I feel I can bring tremendous value to your IT organization.

Bad answers (all real answers from real candidates):

- I don't know.
- You're hiring. I'm desperate.
- You guys are famous. I don't think you'll go out of business.
- I want to climb the corporate ladder.
- I want to work in Silicon Valley.
- I want to see Safeco Field.

Q: **What do you like about Amazon?**

Good answers:

- You have an incredible variety of goods and services, with seemingly limitless market potential.
- I'm an avid user of your site, and I have the following suggestions for improving it...
- I'm overwhelmingly impressed by the technical achievement that's gone into building your website and fulfillment processes. Most people don't realize how hard it is.
- You make your customers happy.

Bad answers:

- I don't know/no answer.
- I heard you guys pay really well.
- I like the Seattle area, and you're there.
- Your stock is going up.

Q: **Why are you leaving your current job?**

Good answers:

- I've worked here for many years and delivered great value to the company, but I feel that I've achieved my potential here, and I need a larger challenge to be able to grow.
- My dot-com startup finally imploded after several rounds of layoffs and failed attempts at securing more funding. We held on for as long as we could, but in the end, there were only ten of us left.
- I enjoy my job, but to be honest, I've always wanted to work at Amazon, and I finally decided I couldn't wait any longer, so I'm applying.
- I feel the company lacks executive-level leadership. I often feel that I could run the company better than they can, because of my in-depth knowledge of the business and competitors.
- Dude, I just graduated.

Bad answers:

- They're firing me.
- My manager's a jerk. A complete kneebiter.
- My 500-person company is doing their first-ever round of layoffs.
- I didn't like the project I was working on.
- I want to do more telecommuting.

Other Culture Fit questions:

- If you were in charge of Amazon, what's the first thing you'd change?
- Do you prefer working on team projects or individual projects? Why?
- What's the hardest project you've ever worked on? Why?
- What's your 1-year career plan?
- What's your 5-year career plan?
- How do you deal with underperformers working on projects with you?
- How do you feel about working long hours during crunch time?
- Who's the worst manager you've ever had? How so?
- What's the worst project you ever worked on? Why?
- How do you estimate how long a task will take you?
- How do you deal with situations where you've got too much work to do?
- How do you deal with ambiguous/missing project requirements?
- What motivates you to work in your profession?
- What do you like the most about your profession?
- What do you dislike the most about your profession?
- If I gave you a million dollars, what would you do?

I'm sure you can think of plenty of others.

Note: it's OK for the candidate to ask for you to clarify questions.

Software Culture Fit

There are various questions you can ask that will give you a feel for how the candidate will fit in your software-engineering environment.

Q: **How much do you comment your code? What's the "right" amount?**

Good answers:

- you shouldn't have to read the code to figure out what it does. If it's even remotely ambiguous or convoluted, you should put in a comment to clarify for the reader.
- other people will be maintaining my code, so I comment it as if I'm seeing it for the first time.
- I tend to add comments to code as I'm going through it.
- It's important to keep the comments up-to-date as the code changes.
- I use automatic API documentation programs like javadoc or Cocoon, so I keep my file and function headers current.

Bad answers:

- you should be able to figure out what code does by reading it
- commenting your code is overrated (or a slippery slope)
- it's too hard to keep the comments up to date, so I don't bother
- if my code was hard to write, it should be hard to understand.
- I hate people who put too many comments in their code.

Anyone who gives one of the bad answers has never worked on a code base with millions of lines of code, and probably won't work out here at Amazon.

Q: **How many lines of code have you written in your career, total?**

Good candidates should be able to make a very rough estimate. Bad candidates will simply fail to answer the question, or even become hostile.

Even if the candidate makes an honest effort at estimating it, if they've only written 5,000 lines of code, they're probably not qualified. My *extremely* rough rule of thumb is:

- 10k lines for a college hire
- 20k lines for a college hire with internship
- at least 50k lines for an SDE2 with 3-5 years of experience
- at least 200k lines for an SDE3

Q: **What text editor do you use in Unix?**

A: It doesn't matter, as long as the candidate can actually name one and claim proficiency with it.

The standard bad answer is "oh, ah, I usually use vi, and I use an IDE in Windows." If they're a good vi user, they use VIM or some other high-powered vi-based editor.

My personal favorite answer is: "I usually don't know which operating system I'm running, because it's just plumbing for Emacs."

Note: None of these questions is, by itself, enough to disqualify a candidate. You add them all up at the end.

Q: **Have you ever worked on a project without source-code control?**

A: Hopefully not, except maybe in school. Candidates should know about source-control systems and ideally have worked with one that allows multiple developers to lock a file. If they haven't, make them explain how it would work.

Q: **Have you ever read (parts of) any of the following books:**

- Design Patterns, by Gamma/Helm/Johnson/Vlissides
- Effective C++ (Scott Meyers)
- C++ Programming Style (Cargill)
- Concurrent Programming in Java (Doug Lea)
- Peopleware (DeMarco and Lister)
- Rapid Development or Code Complete (McConnell)
- any other book on programming style or program design

Another way to ask it is "What is your favorite book on programming or program design?"

The most important takeaway from Software Culture Fit questions is simply this:

"Do you love programming, or do you just do it for a living?"

Project Questions

These questions can apply to school or work projects. Some things to note about this line of questioning:

- It's the same format every time. You can ask this line of questions to anyone and find out if they're smart, even if you're totally unfamiliar with the details of the technology they're using.

- You can use these questions even if you've never read the candidate's resume.

- A really good candidate can B.S. you in this kind of questioning, so make sure to ask at least one technical question *not* related to any of their project work (see the next section).

You don't have to ask all these questions - you can pick any subset of them, and you can jump around. Just start with one of the opening questions and go from there.

Opening #1: "I see that you took a class in XYZ. Do you feel comfortable talking about the work you did in that class?"

Opening #2: "You did an internship for company ABC. Can you tell me about the work you did there?"

Opening #3: "I'd like you to pick a technical project you've done, either for school or for work, that you found particularly interesting and/or challenging, and tell me about it."

If they choose to talk about something you know absolutely nothing about, say this:

"Pretend that I'm completely unfamiliar with this technology or concept, and explain it to me from the ground up."

Then you can get away with asking the stupidest questions and they'll think you're testing their communication skills.

If the candidate can't explain their project to you, then they either don't understand it themselves, or they're not a good communicator. The candidate should be able to explain anything they've done well enough for you to understand it.

General Project Questions

"Why were you doing that project?"
"How big was the team you were working on?"
"What language(s) was it written in?"
"What third-party products did you have to use as part of the project?"
"How much code was there before you started?"
"How many lines of code did the project wind up being?"
"How many lines of code did you write yourself?"
"How long were you working on the project?"
"How long did the project take from beginning to end?"
"What was your role on the project?"
"Tell me about the specific pieces you contributed."
"Tell me about one piece you did in lots of detail."
"Tell me about the pieces other people contributed".
"Who on your team contributed the most?"
"How was the project team organized?"
"What was the hardest problem you faced on the project?"
"Did you use source-code control on the project? Which one?"
"Did you have to document the project? How did you do it?"
"What was the biggest stumbling block or time-waster?"
"What was the biggest mistake you made on the project?"
"Did the project finish on time? If not, why not?"

Extra Questions for Non-School Projects

"Who was the customer for the project?"
"Who wrote the business requirements or functional spec?"
"Why did you build this technology instead of using XYZ?"
"Who else in the organization did your team work with?"
"What was the business justification for doing the project?"
"How much money did it make/save the company?"
"Did the project go into production? Is it in use today?"
- "If so, who's maintaining it now?"
"Why did you leave, or are you leaving?"

Closing Project Questions

"What's the most important thing you learned on the project?"
"If you could do it all over again, what would you do differently?"
"Is there anything else I should know about the project?"
"Is this the kind of work you see yourself doing in the future?"

Project Questions – Observations

These questions are a good way to find out:

- whether the candidate actually did real work on the project, or just sort of "participated" (or just did the easy parts)

- whether the candidate thinks about alternative solutions before jumping in and designing something

- how the candidate works with others on team projects

Great candidates, even at the intern level, will be able to answer business-related questions about their projects. They'll understand the context for doing the project and why the decisions and justifications were made.

Bad candidates will say "I worked on it because my boss or professor told me to", and won't ever have pulled their head up to look around at why they were doing it.

You'll find that some candidates have a track record of "conflict" with their teams, where "people couldn't come to consensus on how to do it". This is sometimes a sign that the candidate is hard to work with.

Technical/Problem-Solving Questions

This question category actually has several sub-categories:

- **broad**, shallow, survey-type questions ("do you know this language?")
- **deep**, specific CS-type questions ("how do hashtables work?")
- **design** questions ("design a parking lot")
- **problem-solving** questions ("find cycles in a linked list")

I'm going to skip the problem-solving questions, since I don't like them in general, and I don't think they're appropriate to ask on the telephone. I could talk for hours about this, but the bottom line is that if you want to ask that kind of question, you'll have to come up with your own. I'm not going to help you.

Survey Questions

Survey questions are rapid-fire, go/no-go questions, in which you ask the candidate one or two things about a particular technology or buzzword on his or her resume.

Why do I do this? Because I've conducted dozens of phone screens in which I spent half an hour going over something very specific, such as C++ programming, only to find in the interview cycle that the candidate didn't know anything else.

I also do it because, quite frankly, candidates often lie on their resumes. I see that more and more these days. Asking one or two pointed questions about a language or technology will give you a feel for how broad the candidate's experience really is.

For survey questions, I usually ask if they're familiar with a particular topic, followed by one or two specific questions about the topic if they claim familiarity.

The following survey questions each a few some sub-questions listed, but you don't have to ask them all.

Do you know C?

- what are the arguments to main()?
- how do you convert a string to an int? (atoi)
- can C accept variable numbers of arguments to functions?
- what is a static variable?
- what does extern mean?
- what's the difference between a definition and a declaration?

Do you know C++?

- what's a reference parameter?
- what's a copy constructor?
- how do you delete an array of objects?
- what does 'protected' mean?
- what's a template?
- what's a static member function?

Do you know how to program in Unix?

- how do you find a file with a given name in your filesystem? ('find')
- what window manager do you use? (twm? enlightenment? fvwm? other?)
- what's the command to see what directory you're in? ('pwd')
- what shell do you prefer? (bash? csh? tcsh? zsh? other?)
- what's the built-in unix documentation called? ('man')
- how do you see what processes are currently running? ('ps')
- what command tells you what CPU and memory are being used? ('top')
- how do you tell how long the computer has been up? ('uptime')

Do you know how to use regular expressions?

- give me a regex to match a 10-digit phone number
- give me a regex to extract the month from the unix 'date' command
- give me a regex to find the third word in a sentence.
- give me a regex to find words in the dictionary with four letter-S's in them

Have you used threads before?

- How do you start a thread?
- What's the difference between a process and a thread?
- Why do you need locks? How are they implemented?
- What happens if you don't use synchronization?
- What happens if you use too much synchronization?
- How does the operating system know when to interrupt a thread?

Are you familiar with object-oriented programming terminology?

- what is a class?
- what is the difference between a class and an object?
- what is instantiation?
- what is a static variable?
- what are the components of a method signature?
- what is encapsulation?
- what is polymorphism?
- what is multiple inheritance?

Are you familiar with Design Patterns?

- describe Singleton pattern
- describe Visitor, Observer or Iterator pattern (any one)
- describe Factory pattern
- what's your favorite pattern? how have you used it in real code?
- if you could add a pattern of your own to the book, what would it be?

Have you used any scripting languages? perl/awk/sed/sh/python/etc?

- how do you declare a variable in that language?
- what kinds of tasks are scripting languages suitable for?
- how do you read lines from a file in your favorite scripting language?
- what's the biggest script you've ever written?
- how do you deploy your code into production?

Have you used a source-code control system?

- which one(s)?
- did it handle multiple locks?
- what's the command syntax for checking out the source tree?

Basic Computer Science Questions

Everyone knows a little computer science, or they should, if they're a programmer. I tend to focus more on the practical nuts-and-bolts stuff, rather than on more theoretical concepts.

Data Structures and Algorithms

Describe how to implement a growable array ("Vector")

- should be backed by a physical array, for performance (not linked list)
- describe the API to the vector (put/get operations)
- tell me how you make it growable
- tell me how the vector decides when to grow the internal array
- what's the performance of growing the array? (copy n elements)
- in C++, how do you enforce only a certain type going into the vector?
- in Java, what must you do when you pull an item out of the array? (cast)
- should the array shrink itself as you remove elements?

Describe how a hashtable works

- how do you use a hashtable? what's it for?
- what are the primary operations on a hashtable? (put/get)
- what's the efficiency of the operations on a hashtable? (put/get/find)
- how do you implement a hashtable?
- what's a hash function?
- what are the properties of a good hash function? (fast, deterministic, random)
- give an example hash function for a string - can be a bad one (e.g. add ascii char values)
- how do you map the hash code into the underlying array? (modulo)
- how do you handle collisions? (linked lists, secondary hashing, linear/quadratic probing)
- what happens as your table fills up? (need to rehash)
- what's the performance of rehashing? (linear)

Describe how a binary tree works

- how do you use one? what kind of data can you store in it?
- what's the basic algorithm for building the tree?
- what are the 2 simplest ways of traversing the tree?
- what is the performance of put/get/search operations?
- what can happen to the tree's performance in the worst-case
- how do you fix this? (balancing)
- describe at least one kind of balanced binary tree. how does it work?
- what does keeping the tree balanced do to the performance?

- how do you build a binary tree in your favorite language?
- describe how non-binary trees work (more than 2 children per node)

Describe one other fancy data structure you've studied or used

- possibilities: btree, trie tree, skip list, splay tree, graph, etc.
- why/when would you use this data structure?
- why not just use a standard one like vector/tree/hashtable?
- how does your data structure work?
- what are its performance characteristics?

Have you used nested/aggregate data structures?

- describe a situation where you'd want to use a hashtable of vectors
- what about a hashtable of hashtables?
- how do you implement aggregated data structures in your favorite language?

Sorting

- how do you sort an array of strings?
- what's the performance of the sorting function you picked?
- what's the best sorting function you know?
- how does it work?
- can you use a binary tree to sort data? how?
- in an unsorted array, what are the odds your item is in the first slot?

OO Design

Most candidates applying for an SDE position (full-time, not an internship) should be able to give satisfactory answers to any of these OO design questions over the phone.

Design a parking lot

- make it able to park cars, motorcycles and handicapped vehicles
- motorcycles should park in moto spots (if any), or else car spots
- permit vehicles can park in handicapped spots (or car spots)
- tell me the classes you'd use to represent the system
- write park() and retrieve() methods and tell me how they work
- how do you make park() and retrieve() perform in constant-time?

Design an algorithm to evaluate an expression tree

- assume someone else has created a tree to represent (2+7)/(6*5)
- what classes would you have in your evaluator?
- write the eval() function

- now do it polymorphically (don't use a switch/case statement)
- *extra credit*: design system to create the expression tree

Design a banking service

- tell me what remote services you'd provide
- design the API
- design a teller machine interface
- how does the teller machine communicate with the bank?
- how do you make it persistent? transactional?

Design a Zoo

Tell me how you'd represent the various components:

- animals
- cages
- dietary restrictions
- animal incompatibilities (who can go together)

Design a chess or checkers game

- how do you represent the pieces? the board? the moves?
- explain how you'd detect valid/legal moves on the board
- how would you make it multiplayer?
- *Extra credit:* tell me how chess playing algorithms work

Threads and Processes

Ask these questions if the person claims profiency with concurrency, multithreading, parallel computing, or OS design:

- what are the pros and cons of threads vs. processes?
- how do you synchronize access to a critical section?
- tell me how to implement a semaphore (or monitor, if they prefer)
- tell me how a fair-share scheduler works
- how do you do deadlock detection?
- how do you assign thread priorities?
- what is priority inversion and why do you use it?
- how do you get the currently-running thread?
- how do you spawn a subprocess programmatically in unix?
- how do threads communicate?
- what are the various ways processes can communicate?
- how do you prevent thread/process starvation?
- give me an example of a good use for a thread
- how much overhead is associated with a thread?

Databases

Ask these questions (or questions like them) if the candidate claims proficiency with relational database design and/or SQL:

- how do you perform an SQL query in your favorite language?
- how does the database return multiple results from a query?
- tell me an SQL query to extract a customer name from a customer table
- tell me an example of an SQL query that does a table join
- how do you have the database sort the results of your query?
- what's a stored procedure? give me an example of when you'd want one.
- what's a bind variable? why do you need them? what's the syntax?
- what are normal forms in data modeling?
- What is denormalization?
- design a data model to implement name/value pairs
- what's a table index? why do you need them? how are they implemented?
- what's a transaction?
- how do you abort a transaction? how do you roll one back?
- how can you do transactions across multiple database instances?
- tell me how you'd implement a join between tables in different databases.

Networking

Ask these kinds of questions if the candidate has had a networking class, or has worked with internet protocols or distributed systems:

- describe the TCP/IP protocol (or the HTTP protocol)
- what's the difference between TCP and UDP?
- describe the sliding window protocol. what's it for?
- what's DNS? how does it work?
- what protocol does the Web use?
- describe how to design a simple RPC system
- how does a socket work? What are the parameters to opening one?
- how do you tell if data has arrived on an open socket?
- how do you do non-blocking I/O on a socket?
- describe any higher-level mechanism for remote-method invocation
- (java RMI, CORBA, publish/subscribe, XML over HTTP, SOAP, etc.)
- what unix command can you use to see if a host is up? (ping)