Phases of Interviews

Each round of interview is usually split into three phases, as shown in Figure 1-2. The first phase is the behavioral interview, in which interviewers examine candidates' experience while referring to their résumés. The second phase is the technical interview when it is highly possible for a candidate to be asked to solve some coding interview problems. Finally, the candidate is given time to ask a few questions.

Figure 1-2. *Three phases of a round of interview*

Behavior Interview

The first five to ten minutes of a round of interview is used for becoming acquainted. Usually, this is time for the behavioral interview, and no difficult technical questions are asked. Interviewers look for someone who would be a good fit for the job in terms of technical skills as well as personality. A person who is too timid might not fit well into an environment where he or she needs to be vocal. Interviewers also look for enthusiasm and excitement. If candidates are not excited about the position, they may not be motivated to contribute, even if they are a strong technical fit.

Most interviews begin with candidates' introducing themselves. A candidate usually doesn't need to spend a lot of time introducing his or her main study and work experiences because interviewers have seen his or her résumé which contains detailed information. However, if an interviewer feels interested in a project the candidate has worked on, he or she may ask several questions on that subject in the introductory phase.

Project Experience

After a candidate has introduced him- or herself, interviewers may follow up with some questions on interesting projects listed on his or her résumé. It is recommended to use the STAR pattern to describe each project both on your résumé and during interviews (Figure 1-3).

- *Situation:* Concise information about the project background, including the size, features, and target customers of the project.
- *Task:* Concrete tasks should be listed when describing a big project. Please notice the difference between "taking part in a project" and "taking charge of a project." When candidates mentioned they have taken charge of a project, it is highly possible for them to be asked about the overall architectural design, core algorithms, and team collaboration. These questions would be very difficult to answer if the candidates only joined a team and wrote dozens of lines of code. Candidates should be honest during interviews. Reference checks will also query claims made on résumés.
- *Action:* Detailed information should be covered about how to finish tasks. If the task was architectural design, what were the requirements and how were they fulfilled? If the task was to implement a feature, what technologies were applied on which platforms? If it was to test, was it tested automatically or manually, with black boxes or white boxes?
- *Result:* Data, especially numbers, about your contribution should be listed explicitly. If the task is to implement features, how many features have been delivered in time? If the task is to maintain an existing system, how many bugs have been fixed?

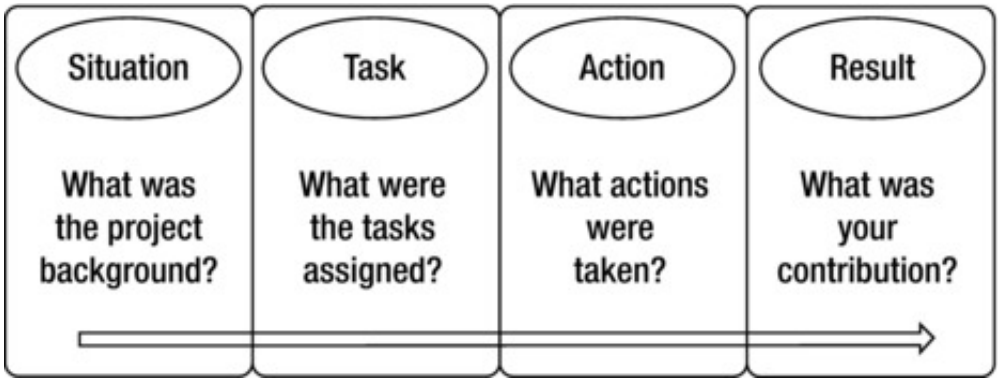

Figure 1-3. *A STAR pattern describes project experiences on résumés and during interviews.*

Let's look at an example of the STAR patter in use. I usually describe my experience working on the Microsoft Winforms team in the following terms:

Winforms is a mature UI platform in Microsoft .NET (*Situation*). I mainly focused on maintenance and on implementing a few new features (*Task*). For the new features, I implemented new UI styles on Winforms controls in C# in order to make them look consistent between Windows XP and Windows 7. I tried to debug most of the reproducible issues we had with Visual Studio and employed WinDbg to analyze dump files (*Action*). I fixed more than 200 bugs in those two years (*Result*).

Interviewers may follow up with a few questions if the information you supplied in these four categories has not been described clearly. Additionally, interviewers are also interested in the candidates' answers to the following questions:

- What was the most difficult issue in the project? How did you solve it?
- What did you learn from the project?

- When did you conflict with other team members (developers, testers, UI designers, or project managers)? How did you eliminate these conflicts?

It is strongly recommended that candidates prepare answers to each of the questions above when they write their résumés. The more time they spend preparing, the more confident they will be during interviews.

Tip When describing a project either on a résumé or during an interview, candidates should be concise regarding project background, but they should provide detailed information about their own tasks, actions, and contributions.

Technical Skills

Besides project experiences, technical skills are also a key element that interviewers pay close attention to on candidates' résumés. Candidates should be honest in describing the proficiency level of their skills. Only when candidates feel confident that they are capable of solving most of the problems in a certain domain, should they declare themselves experts. Interviewers have higher expectation of candidates who claim to be experts and ask them more difficult questions. It is very disappointing when you cannot meet these expectations.

For instance, I interviewed a candidate who declared himself an expert on C++, but he could not answer questions about the initialization order of parameters in constructor functions.

Tip Candidates should be honest when they are describing their project experiences and technical skills.

"Why Do You Want to Change Your Current Job?"

If a candidate already has worked for a few years, it is highly likely for him or her to answer questions regarding his or her reasons for wanting to change jobs. Because everyone has his or her own reasons for wanting a new position, there are no standard answers. However, not all answers are appropriate for interviews.

Candidates should avoid complaining because complaining reveals passive emotions, and passive emotions are usually infectious in a team. When a candidate complains a lot, his or her interviewer worries that he or she might become the source of passive emotions and affect the morale of the team as a whole.

Complaints often arise from four categories. These complaints should be avoided during interviews:

- *My boss is rigorous with high standards and strict requirements.* If an interviewer is the manager of the hiring position and she hears such an answer, she might wonder if she would be the next rigorous boss after she hires the complaining candidate.
- *My teammates are not affable and friendly.* If a candidate believes all of his or her teammates are not easy to get along with, the candidate may be the one who is difficult to work with.
- *We work overtime too often.* All companies look for diligent workers, and it is not rare to work overtime in software and Internet companies. If the hiring position also requires overtime, this response may disqualify the candidate.
- *My salary is too low.* Having a low salary is indeed the real reason many candidates want to change jobs. However, it is not a suitable time to discuss salary requirements during the technical interviews. The only purpose for interviews is to get an offer. Candidates can discuss salary packages with the recruiters when they pass the technical interviews.

A recommended answer on job hopping is to use the job you are applying for as the model job: "My ideal job is to be working in the position you are hiring for because I am looking for a more challenging job. I do not have much passion for aspects of my current job, and I would like to take on a more fulfilling position." Some detailed reasons should be given as to why you feel unmotivated on your current job and why you are interested in the new position.

After I got on board, one of my interviewers at Microsoft told me that my answer was impressive. My previous job at Autodesk was to develop new features for Civil 3D, which is well-known software for civil engineering. I had to learn more civil engineering before I got promoted, but I was not interested in the domain knowledge such as earthwork calculation, corridor design, and so on. Therefore, I was looking for opportunities outside the company.

Technical Interview

After interviewers get candidates' background information through the behavioral interviews, they move on to the technical interview. Technical questions require about 40 to 50 minutes if the overall interview time is an hour. This is the most important phase of the whole interview process.

Interviewers are generally interested in skills in five categories:

- Basic programming knowledge, including understanding of programming languages, data structures, and algorithms
- Abilities to write clean, complete, and robust code
- Capabilities to analyze and solve complex problems
- Abilities to improve time and space efficiencies
- Skills involving communication, learning, divergent thinking, and so on.

Candidates should be well prepared before the interview and master knowledge of programming languages, common data structures, and algorithms. If the coding interview questions are simple, candidates should pay attention to details to write complete and robust code. If questions are difficult, they may try to simplify problems with figures and examples, and by dividing problems into manageable subproblems in order to get clear solutions before coding. Moreover, they should try their best to improve time and space performance. Candidates can ask interviewers to clarify their requirements in order to demonstrate communication skills. It is not difficult to get an offer if a candidate performs well on these factors.

Basic Programming Knowledge

Having abundant programming knowledge is the key to being an outstanding developer, so it is the first aptitude to be examined during the interview. It includes understanding of programming languages, data structures, and algorithms.

First, every programmer should be proficient in at least one or two programming languages. Interviewers examine the proficiency level of programming languages via coding and follow-up questions. Take C++ as an example. If there is a pointer parameter in a candidate's source code, he or she might be asked whether the parameter should be marked as `const`, and what the differences are if `const` is placed before or after an asterisk symbol (*).

Second, many interview questions focus on data structures. Lists, trees, stacks, and queues appear frequently in interviews, so candidates should be familiar with their operations. For example, candidates should write bug-free code quickly to insert and delete nodes of lists, as well as traverse binary trees iteratively and recursively with pre-order, in-order, and post-order algorithms.

Last but not least, algorithms are focal points in many interviews. Candidates should be familiar with the differences between various searching and sorting algorithms as well as scenarios suitable for each algorithm. There are many interview questions that are actually utilizations of binary search, merge sort, and quick sort algorithms. For example, interview questions "Minimum of Rotated Array" (Question 27) and "Times of Occurrences in Sorted Array" (Question 83) are about the binary search algorithm, and the problem "Reversed Pairs in an Array" (Question 81) is essentially about the merge sort algorithm. There are also many interview problems concerning dynamic programming and greedy algorithms.

Four of the most popular programming languages, C, C++, C#, and Java are covered in Chapter 2. The most common data structures are discussed in Chapter 3, and algorithms are discussed in Chapter 4.

Clean, Complete, and Robust Code

Many candidates are confused and ask themselves, "Why wasn't I hired even though the interview questions seemed simple to me?" There are many reasons why people fail, and the most common one is that there are some problems remaining in their solutions or written code. That is to say, they have to improve their code quality.

The first standard of code quality is readability. If candidates are asked to write code on paper or white boards, they must write neatly and cleanly. Additionally, readability is improved if code is written with reasonable variable names and logical indentation.

The second standard is completeness. Many interviewers examine quality through boundary checking and special inputs (such as `NULL` pointers and empty strings). There are lots of candidates who fail their interviews because their code only fulfills the basic functional requirement.

Let's take one of the most popular interview questions in Microsoft as an example: How would you convert a string into an integer? (See Listing 1-1.) This question seems very simple, and some candidates can finish writing code within three minutes.

Listing 1-1. *C Code to Convert a String to an Integer*

```
int StrToInt(char* string) {
    int number = 0;
    while(*string != 0) {
        number = number * 10 + *string - '0';
        ++string;
    }

    return number;
}
```

Do you also think this problem is quite easy after reading the code above? If you think so, it is highly possible that you will be rejected by Microsoft.

The simpler the question is, the higher expectations an interviewer has. The problem above is simple, so interviewers expect candidates to solve it completely. Besides basic functional requests, boundary conditions and error handling should be considered. Converting a string into an integer is the basic functional request to be fulfilled. Additionally, candidates should pay attention to more cases, including the negative and positive symbols, the minimal and maximal integers, and overflow. The code is also expected to handle cases when the input string is not numeric, with non-digit characters. When we take all of these cases into consideration, it is not a simple problem anymore.

Besides incomplete solution and code, another intolerable mistake from an interviewer's perspective is that code is not robust enough. If we scrutinize the code above carefully, we notice that it crashes when the input string is a `NULL` pointer. It would be a disaster if such code were integrated into a real software system.

Not all issues related to robustness are so obvious. Let's take another popular problem as an example: how to get the k^{th} node from the tail of a list. Many candidates read its solution with two pointers on the Internet. The first pointer moves k-1 steps, and then two pointers move together. When the first pointer reaches the tail node of the list, the second one reaches the k^{th} node from the tail. They feel lucky and write the code in Listing 1-2 with much confidence.

Listing 1-2. *C++ Code to Get the k^{th} Node from Tail*

```
ListNode* FindKthToTail(ListNode* pListHead, unsigned int k) {
    if(pListHead == NULL)
        return NULL;

    ListNode *pAhead = pListHead;
    ListNode *pBehind = NULL;
    for(unsigned int i = 0; i < k - 1; ++ i) {
        pAhead = pAhead->m_pNext;
}

    pBehind = pListHead;
    while(pAhead->m_pNext != NULL) {
        pAhead = pAhead->m_pNext;
        pBehind = pBehind->m_pNext;
}

    return pBehind;
}
```

The candidate who writes the previous code feels more confident when he or she finds that the `NULL` pointer is handled and, consequently, believes he or she will definitely be hired. Unfortunately, a rejection letter might be received a few days later because there are still two serious issues left: (1) When the number of nodes in a list is less than *k*, it crashes; (2) When the input *k* is zero, it crashes.

The best approach to solving this kind of problems is to figure out some test cases before coding. Candidates may write complete code only if all possible inputs have been considered. It is not a good strategy to ask interviewers to check their code immediately after they finish writing the code. They should execute their code in their minds first and only hand the code to interviewers after they are sure it gets expected results for all test cases.

We will discuss strategies to improve code quality in more detail in Chapter 5.

Tip Besides basic functional requirements, interviewers expect candidates to handle boundary conditions, special inputs (such as `NULL` pointers and empty strings), and errors.

Clear Thoughts about Solutions

Candidates cannot solve complex problems without clear thinking. Sometimes interviewers ask very difficult questions that they do not even expect candidates to solve in less than an hour. What interviewers are interested in are the candidates' thinking processes in such situations, rather than complete and correct answers. Usually, interviewers dislike candidates who write code in haste, with lots of bugs in it.

There are a few methods to help candidates to formulate clear ideas. First, candidates can employ some examples to help themselves to analyze problems. Simulating operations with one or two examples may uncover the hidden rules. Second, figures are helpful to visualize abstract data structures and algorithms. Many problems related to lists and binary trees might become easier if they are visualized with figures. Another approach is to divide a complicated problem into multiple manageable pieces and then solve them one at a time. Many recursive solutions, such as divide-and-conquer algorithms, as well as dynamic programming algorithms, are all essentially utilization of division.

For example, it is quite a complex problem to convert a binary search tree into a sorted doubly linked list. When confronted with such a problem during interviews, candidates may draw some figures about one or two sample binary search trees and their corresponding sorted doubly linked lists in order to visualize the relationship between them. They can also try to divide a binary tree into three parts: the root node, the left subtree, and the right subtree. After the left and right subtrees are converted to linked lists, they are connected with the root node to form a sorted linked list, and the problem is solved. More details about this problem are available in the section *Binary Search Trees and Double-Linked Lists*.

We will discuss how to solve difficult problems with figures, examples, and division in more detail in Chapter 6.

Tip Three tools are available to analyze and solve complicated problems: figures, examples, and division.

Time and Space Efficiency

Outstanding developers pay a lot of attention to time and space consumption and have the passion needed to continue improving performance of their own code. When there are multiple solutions for a problem, interviewers always expect the best one. If interviewers point out that there are better solutions, candidates should try their best to find approaches to improving time and space performance, demonstrating his or her enthusiasm to pursue excellence, which is an essential spirit to have as an outstanding developer.

The first thing candidates should understand is how to analyze time and space efficiencies. Various implementations of the same algorithm may result in dramatic performance distinctions, so it is important to analyze performance of an algorithm and its implementation. Take the calculation of the Fibonacci Sequence as an example. The classical solution is based on the recursive equation $f(n) = f(n\text{-}1) + f(n\text{-}2)$. It is not difficult to find out the time complexity increases exponentially since there are lots of duplicated calculations. However, the complexity will reduce to $O(n)$ if it is calculated iteratively. First of all, $f(1)$ and $f(2)$ are calculated, and then $f(3)$ is based on $f(1)$ and $f(2)$; $f(4)$ is get based on $f(2)$ and $f(3)$. The sequence continues until $f(n)$ is calculated in a loop. Please refer to the section *Fibonacci Sequence* for more details.

Candidates have to master pros and cons of each data structure and be able to choose the most suitable one to improve performance. For example, it seems that multiple types of data structures are available to get the median of a stream, including arrays, lists, balanced binary trees, and heaps. After analyzing the characteristics of each data type, we find that the best choice is to utilize two heaps—a maximal heap and a minimal heap (section *Median in Stream*).

Candidates should also be proficient in common algorithms. The most popular algorithms in interviews are about search and sort. It costs $O(n)$ time to scan an array sequentially. However, it is reduced to $O(\log n)$ with the binary search algorithm if an array is sorted. The problem "Maximal Number in a Unimodal Array" (Question 28) and "Times of Occurrences in a Sorted Array" (Question 83) are both solved based on the binary search algorithm. The quicksort algorithm is widely used for other problems besides sorting. The `Partition` function in quicksort can be used to get the k^{th} maximal number out of n numbers and solve the problem "Majority in an Array" (Question 29) and "Minimal *k* Numbers" (Question 70).

More tips to improve time and space performance are illustrated in Chapter 7.

Soft Skills

Besides programming skills and technical capabilities, candidates should also show their soft skills, such as communication skills and learning abilities.

Since software systems have become more and more complex, it is more and more difficult for a single engineer to develop successful software. Teams, as well as communication among team members, are very important nowadays. Descriptions of project experience and algorithms with clear logical reasoning and conclusions are demonstrations of communication skills. Moreover, candidates reveal their consciousness about team cooperation through their tone and facial expressions. In general, interviewers dislike arrogant candidates who despise their teammates.

Developers should have strong learning spirits and capabilities because knowledge and technologies in the IT industry are replaced by more advanced ones at a frequent pace. Therefore, learning ability is also an important factor during interviews. There are two strategies for interviewers to examine learning ability. The first strategy is to ask candidates what books they have read recently and what new knowledge and skills they have learned. The other one is to raise some new concepts and observe how much time candidates need to understand them. For example, an interviewer asks to get the 1500^{th} ugly number, which is a new concept for many candidates. A candidate with strong learning ability usually follows the process of asking questions to the interviewer, thinking to him- or herself based on the interviewer's clarification, asking questions again, and so on, until he or she finally finds the rules of ugly numbers and solves the problem.

The knowledge migration skill is a special capacity to learn new things. It is easier to learn new technologies and solve new problems if we can apply our own knowledge to new domains. Interviewers often ask a simple question at first and then follow up with a difficult one. In such situations, they expect candidates to get some hint from the solution of the simple question. For example, an interviewer asks how to calculate the Fibonacci Sequence and then asks a question about jumping frogs: There is a stair with *n* levels. A frog can jump up one level or two levels at each step on the stair. How many possible ways are available for this frog to jump from the bottom of the stair to the top? If candidates have strong skills to borrow ideas from experiences, they may find out that it is actually a utilization of the Fibonacci Sequence.

Some interview problems also require mathematical modeling skills. These kinds of questions are abstracted from our daily life. In order to solve them, candidates have to discover the hidden rules and choose appropriate data structures and algorithms. For instance, the problem "Minimal Number of Moves to Sort Cards" (Question 97) is closely related to the classical problem "Longest Increasing Sequence in an Array," which can be solved with dynamic programming algorithms.

Many interviewers are interested in a candidate's divergent thinking and innovation capabilities. They disallow candidates to use common solutions and expect candidates to explore creative approaches to analyzing and solving problems. For example, it needs divergent thinking skills to implement addition,

subtraction, multiplication, and division without +, -, * and / operators.

Each of these soft skills is discussed in detail in Chapter 8 with several sample interview problems.

Q/A Time

Five to ten minutes before the end of an interview, a candidate is invited to ask a few questions. The quality of the candidate's questions also impacts the interview result because it reflects the candidate's communication skills. Usually, a person who is good at asking questions has strong communication skills.

Candidates should ask at least one or two questions to show their interest in the hiring company and position. Sometimes candidates become exhausted due to many hard interview problems, so it is difficult for them to think of some interesting questions right away. Therefore, it is suggested to prepare some questions in advance.

Not all questions are suitable to ask during technical interviews. One type of inappropriate questions is irrelevant to the hiring position. For example, the question about the company strategy in the following five years is appropriate only for a CTO candidate when the interviewer is a CEO. It is too far away from interests of a junior developer. Moreover, it is also very difficult for the interviewer to answer if he or she is an ordinary engineer.

It is not recommended that you ask about salary during technical interviews. It is not going to impress interviewers if they notice the most important issue in a candidate's mind is salary. Please leave this kind of question to recruiters afterward.

It is not suggested that you ask for the interview result immediately either: "Do you thing I am qualified enough?" A single interviewer cannot give an answer because the final result is the comprehensive consideration of all interviewers. In some companies, such as Google, all hiring decisions are made by a specific committee. Moreover, such a question is evidence that the candidate who asks it lacks self-evaluation abilities.

Recommended questions are about hiring positions and projects. It is not easy to ask this type of question, so candidates should be prepared in order to collect the related background information. There are two methods available: The first one is to collect information on the Internet. Candidates should be familiar with the history of the company, its business domain, position requirements, and so on. The other method is to pay attention to interviewers' words. Most interviewers describe the hiring position and project progress before asking questions. Candidates may come up with some questions accordingly.

When I was interviewed at Cisco, the manager of the hiring position was the last on-site interviewer. She told me that her team was tasked with the responsibility to develop a platform to test network equipment for Cisco. I asked several questions based on this information: Is it necessary for her team members to take part in network equipment tests? What knowledge is required about hardware tests as well as network equipment?