Software Engineering Career Track Mock Interview Preparation Guide

Student Version
May, 2021

Table of Contents

Mock Interview Introduction

This guide is designed to train you to prepare for and succeed in mock interviews in the Software Engineering Career Track. This will help you assess your performance. The purpose of this guide is to teach you the following:

- The common basic types of interviews and thus mock interviews you will do;
- The timing and flow of a Springboard mock interview;
- Advice on taking feedback during and/or after the interview;
- The number of and type of questions or tasks that are ask in an interview;
- The types of questions and tasks that generate the most useful information.

Students will be required to pass 4 mock interviews throughout the Software Engineering Career Track. The Project Walkthrough, Coding, and Whiteboard Interviews will last 60 minutes each. The Behavioral Interview takes 30 minutes, there is a seperate guide for the Behavioral interview later in this unit. You will have three chances to pass each interview.

Interview Types and Expectations

The Software Engineering Track interviews covered in this guide include:

- Project walkthrough/general technical interview
- Coding
- Whiteboarding

Interview types vary by company and role. The purpose of the Springboard mock interview program is to familiarize you with the most common interview types in the field of

Software Engineering. Depending on job level and the particulars of a given role, Software Engineering candidates may also see additional types beyond those covered in this manual. For example, a manager candidate will have an interview that assesses people management and recruiting, while a candidate for a role at a company with scale and real-time performance constraints may have interviews that focus on system design. We will not be preparing students for interviews associated with specialized roles, but rather **general entry-level positions**.

The first interview type this guide covers is a combination of technical project walkthrough and general technical:

In the **technical project walkthrough** portion, the candidate will demonstrate their experience building full-stack applications. The scope includes React component design, API design, and Object Oriented design as well as general knowledge of web development. It is also an opportunity to see how the candidate communicates, both technically and non-technically.

In the project walkthrough, covering the full scope of a project's lifecycle is important. At some companies, this interview will also assess how well the candidate combines software with product and intuition (e.g. "How would you use software to improve X?", "Why did you use X technology or Y database?").

The **general technical interview** portion assesses a candidate's ability to talk about software engineering concepts and projects that the candidate is knowledgeable about. While the project walkthrough is about past work, this interview can force the candidate to approach a totally new problem with questions around adding additional functionality to an existing project. Questions around why the candidate made certain technical and/or design decisions in the project are also commonplace. Besides assessing the breadth of the candidate's knowledge, this interview is also a chance for the interviewer to see if they would enjoy collaborating with the candidate as a teammate.

The **general technical interview** portion assesses a candidate's ability to talk about software engineering concepts and projects that the candidate is knowledgeable about. While the project walkthrough is about past work, this interview can force the candidate to approach a totally new problem with questions around adding additional functionality to an existing project. Questions around why the candidate made certain technical and/or design decisions in the project are also commonplace. Besides assessing the breadth of the candidate's knowledge, this interview is also a chance for the interviewer to see if they would enjoy collaborating with the candidate as a teammate.

Note: In a real life interview situation, the technical project walkthrough and the general technical interview are usually two separate interviews. For the purposes of the Springboard mock interviews, these two interviews will be combined into one mock interview.

The **coding** interview should already be familiar to experienced software engineers; typically, software engineering candidates receive coding interviews via a shared text editor, coding platform, or Google Docs. This interview assesses the candidate's problem solving ability, knowledge of algorithms and data structures, and coding ability — including testing, debugging, and code quality.

The **whiteboarding** interview should also already be familiar to experienced software engineers; typically this is done in person or on Google Doc where candidates do not have an environment to run the code. This interview assesses the candidate's problem solving ability, knowledge of algorithms and data structures, and coding ability, focusing on pseudocode and problem solving.

Finally, the **non-technical** interview is essentially a **behavioral** interview. The purpose of this interview is to check that the candidate's career trajectory and goals align with the position. It is also a way to determine whether or not the candidate is a *cultural fit* for a company. This type of interview is typically conducted by a manager at a company, often the hiring manager. While it can feel like an easier interview for the candidate, it is fairly difficult for the interviewer because the purpose of the interviewer is to gauge subjective factors about the candidate. Candidates should research the culture and values of a company in order to prepare for this interview. **Preparation for this interview is covered in a seperate guide.**

Springboard Completion Requirement

In order to be eligible for our job guarantee, you'll need to pass all four mock interviews: project walkthrough/general technical interview, coding, whiteboarding, plus a non-technical behavioral (covered in a separate guide in the curriculum). You'll have **three chances** to pass each interview. **If you don't pass one or more of the interviews, you'll still be able to complete the course but will no longer be eligible for the job guarantee.**

Each mock interview type has a dedicated section in this guide:

Interview Type	Minutes	Description
Project Walkthrough and General Technical	60	A **mostly** technical discussion of a project the candidate has built and a set of technical conceptual questions around full-stack software engineering
Coding	60	A **traditional algorithms and coding** interview that is prevalent among software engineering applications

Whiteboarding	60	A traditional algorithms and coding interview without a text editor
Non-Technical / Behavioral	30	A discussion about the candidates experience, values and career goals and how they may fit within an organization

Each mock interview will be graded on a **4 level scale** split across key criteria that check for professional software engineering competencies in line with business hireability and market expectations. The grading criteria is:

- Level 1: Not yet meeting expectations
- Level 2: Approaching expectations
- Level 3: Meeting expectations
- Level 4: Exceeding expectations

General Technical And Project Walkthrough Interview

Topic	Time Allocated
Introduction	2 Minutes
Project Walkthrough	30 Minutes
General Technical Questions	20 Minutes
Feedback and Conclusion	8 Minutes

Interview Flow

The interviewer will spend the first few minutes introducing themselves and providing some context for the interview. The interviewer should spend **50 minutes** discussing your project and asking general technical questions — **30 minutes** on the project walkthrough and **20 minutes** on general technical interview questions. The interviewer may ask you for clarification and why you made certain decisions. If you give a response that makes an assumption, you may be asked about the tradeoffs involved in your choice. You may be asked if you would have done the same thing if you did the project again. The **final 8 minutes** of the interview will be dedicated to giving you feedback on your interview.

Introduction - 2 minutes

This will be a short introduction where the mock interviewer introduce themselves to you and provides an overview of the structure of the programming interview. The mock interviewer will check if you have any questions before starting. They should also check if you can share your screen before formally commencing the interview.

Project Walkthrough and Areas to Cover - 30 minutes

The project walkthrough is a common interview type for Software Engineers. The project walkthrough is a chance for you to go deep on a subject you are knowledgable about. This is often a more useful signal to hiring managers than task-specific expertise; it correlates well with general software understanding and ability. Experienced candidates should discuss something they worked on in a previous job; candidates who haven't worked in industry before can focus on a school or capstone project. The project walkthrough should cover the problem to be solved, data collection (via scraping, APIs, user input), technical decision making (why a certain language / tech was used). The walkthrough should also include existing features and functionality as well as technical decision making around the what and why for these features.

While this interview will be primarily technical, the interviewer can also use it to get sense of whether you can communicate well with a non-technical audience, such as a project manager.

Stages of the project to be assessed:

1. **Purpose, motivation and description:** What is your project? Why was it important? You should be able to thoroughly describe what problem you tried to solve and why the problem was important, even if there was no real client. You should be able to describe features, todos, and tradeoffs in technical decision making.
2. **Data model:** What kind of database was used? Why? How was the data obtained? Was this via web scraping, an API, or user input? How is the data model structured? Is the data model sufficient for the needs of the application? The goal here is to clearly identify where the data originated and how it is being thought about. Wherever the data came from, the interviewer will ask follow-up questions to probe your depth of understanding. Some examples:
    (1) What kind of database relationships did you use in your model?
    (2) How was the data structured into tables?

(3) What are some interesting queries or database operations they've included in this application?
(4) What database systems did you work with? Was this the best choice? What else would have worked and what would the tradeoffs be?
(5) What are some challenges you faced with this design?

You may struggle to answer some of these, as they get into the realm of database administration. It is a good idea to be at least be aware of some of these areas.

3. **API or routing design:** If there is an API built or routing set up on the backend, are RESTful routing conventions followed? How is the backend structured and how are routes declared? Are the correct HTTP verbs being used for each of the routes? The interviewer will ask follow-up questions to probe the your depth of understanding. Some examples:
   (1) How did your decide to structure the routes?
   (2) Did you follow RESTful routing in this application?
   (3) How are you securing your API?
   (4) What are some challenges you faced with this design?
4. **Front-end:** You should be able to specify how the front-end is structured with a framework. For example, you may discuss the component hierarchy and why their design works in a certain way. Some examples of questions you may be asked are:
   (1) How did you decide what the component hierarchy is in your application?
   (2) How are you managing global or app state (like a current user) in your application?
   (3) Why did you decide to use a front-end framework? How did that help your development process?
5. **Additional Features:** You should be able to discuss what makes this project different from a standard CRUD app. The candidate should also be able to describe any interesting features and pieces of functionality that make this app interesting. Here are some sample questions:
   (1) What are some interesting features that you added to your application?
   (2) What are some of the biggest challenges you had with this application?
   (3) If you had some more time what would you add next? Why?
6. **Styling:** You should be able to explain how the application is styled and it should look presentable. You should be able to defend their decision around using a CSS framework and should have a responsive design. Some sample questions:
   (1) Is their application responsive? How did they make that happen?
   (2) What kinds of CSS architecture or styling patterns are they using?
   (3) What kinds of CSS reuse do they have in this application?
7. **Testing:** You should describe what parts of the application are tested either by coverage or explaining the test suite. Testing is essential in projects so you should discuss your testing philosophies, what parts of the application are tested and whether unit, integration, and/or end-to-end tests are used.
   Some sample questions:
   (1) What kinds of tests exist in this application?
   (2) Did you use TDD or another process to write your tests?
   (3) What is your testing philosophy both on the front-end and backend?

8. **Deployment and next steps:** This phase is overlooked by many candidates, especially those who don't have industry experience. The goal of this final section of the walkthrough is to see how the project migrated to a production environment with a URL that can be visited from any browser. Here are some sample questions:
    (1) How did you deploy the application? Is this one app or two apps deployed separately?
    (2) How are you handling environment variables?
    (3) Are there any performance or scaling considerations for this application?

The project walkthrough interview type also addresses the following behavioral characteristics which should be noted in interviewer feedback, or discussed with the candidate:

- **Clear and effective communication**
- **Technical breadth and depth**
- **Patience and cooperation with the interviewer's time**
- **Teamwork (if applicable):**
- **Excitement about software engineering** (make sure you sound excited about software development!)

Score	Observational Behaviours
Exceeding expectations (Score = 4)	● Candidates can clearly walk through all eight steps/stages and answer questions thoroughly around front-end and back-end design. Candidates should be able to answer any question about code or application design in the project, in detail. For the specific parts: ○ Data Model - Candidates can explain in detail why tables are organized in a certain way and why relationships are modeled in a certain fashion. ○ API design - If the API is a REST API, candidates can demonstrate if they have correctly built a REST API. Candidate can explain in detail why they have designed an API a certain way. ○ Front-end - If React is used, candidates can explain the component hierarchy and why the application is structured in that fashion. If a framework is not used, candidates can explain why they chose not to use a framework and can explain how their code is structured properly. If the candidate has used Python/Flask or a server side templating language - are their templates organized properly and are they making use of best practices (template inheritance, not repeating HTML)

	○ Additional features - Candidates must demonstrate that they have built something that is a differentiator from other projects (more than just a CRUD app) ● The application has a thorough readme detailing the technologies used, how to start the application, and how to run the tests ● The application is styled nicely and presentable ● The application contains unit and integration tests and is deployed *Example Question- How are you ensuring that users who are not logged in do not visit your admin page or maliciously update other users?* *Answer: "I have a few different checks to make sure that only the logged in user or an admin can update a profile. I have logic on the backend which takes an incoming request (middleware in Express or @app.before_request in Flask) and checks the authorization information (a cookie or JWT) to search for a user in the database. If there is no authorization information, I respond with a 401 status code with a message of "Unauthorized". If there is authorization information and the user does exist in the database, I then check to see if the user who is making the request is an admin or is the correct user based on URL params. This code exists in my middleware folder and I have written a few unit tests to mock the request and response and ensure that my error handling is correct. I've also added a few integration tests to make sure that when authorization information is passed in, I am reading it correctly,."* *Example Question 2- How are you testing your application?* *Answer: "My application contains unit test and integration tests on the backend. I am using coverage to make sure that a meaningful portion of the codebase is tested and I have tested my models and routes separately. On the front-end I have unit and integration tests as well to make sure that the user interface is what is expected. For specific pieces of functionality that are more difficult to test, I have mocked that functionality. Since this is an Express application on the back-end, I am using supertest for integration tests and Jest for unit testing and for running my tests. You can see in the README.md that in order to run the tests, you'll need to make sure the test database is created and then you can run npm test to run the tests."*

Meeting expectations **(Score = 3)**	• Candidate can answer 75% of the questions highlighted in the above section (Data Model, API design Front end, additional features) • The application has a readme with information about the project • The application is deployed and styled • The application does not contain tests *Example Question- How are you ensuring that users who are not logged in do not visit your admin page or maliciously update other users?* *Example answer : "I am taking a JWT and validating that it is correct using the jsonwebtoken module. I am using middleware (or @app.before_request in Flask) to make sure that there is a token and that it is valid. I am then going to the database to see if the user is an admin and if the data is correct."* *Example Question 2- How are you testing your application?* *Answer: "My application has a few unit and integration tests on the backend to make sure that data received is what is expected. I am using Express so I am using Jest and Supertest to test my routes and models."*
Approaching expectations **(Score = 2)**	• Candidate can not thoroughly answer questions around technical decision making in their application • The application is not deployed and styled *Example Question- How are you ensuring that users who are not logged in do not visit your admin page or maliciously update other users?* *Example answer: "I am using authentication and authorization to protect my routes. If a user is not logged in they can not reach an endpoint in my application. I am using tokens to do this."* *Example Question 2- How are you testing your application?* *Answer: "I've clicked through all the parts of certain pages and done some manually testing to make sure that the application works as expected"*

Not yet meeting expectations (Score = 1)	• Candidate can not answer questions around technical decision making. The application is not deployed,

General Technical Interview - 20 minutes

The **general technical interview** typically consists of the interviewer asking the candidate to answer general technical questions to show an understanding of software engineering concepts and technologies.

This is a list of areas to cover during this interview. The following topics will be addressed at least minimally in any interview. You may ask be interviewed on specific topics if you are focused on certain technologies. **If you wish to be tested on Python you must specifically request this, otherwise you will only be questioned on JavaScript.**

- How the web works
- HTML
- CSS
- JavaScript
- SQL
- Node or Python
- Express or Flask
- React

Questions in this section will:

- Have a correct answer and not be opinionated or open ended.
- Have an answer that can be given in 3-5 minutes.

Score	Observational Behaviours
Exceeding expectations (Score = 4)	• Provides a clear and concise answer to >85% of the questions provided • Provides examples of when certain technologies or concepts are used *Example Question- What is the difference between GET vs POST? Answer: "They are both HTTP verbs that accompany an HTTP request. They can both be used when submitting forms, making AJAX requests and server side requests. GET requests are safe and idempotent, data*

	is stored in the query string, and GET requests are cached by the browser. GET requests are also made when clicking on links or visiting a URL in the browser bar. POST requests are not safe and are not idempotent, they are not cached by the browser and data is sent in the body of the request." *"I've used GET requests when building forms and making HTTP requests with axios from the browser or server side. I usually use GET when making a request to filter data, sort data, or search for something. I've used POST requests for modifying data on the server - this might include registering for a website, charging a credit card, or logging into my account.* *Second Example - What are some differences between cookies and localStorage?* *"Cookies and localStorage are both domain specific and stored on the browser, but localStorage can contain much more information than cookies. LocalStorage can only be accessed using JavaScript in the browser, whereas cookies can be read/accessed on the client and on the server. Cookies can also be created on the server (most commonly) and in the browser as well. With HTTP, a cookie from a specific domain will be sent to the server on every single HTTP request in a header called cookie and when the server creates a cookie it will be sent to the browser in a header called Set-Cookie". Cookies have configuration options for how long the cookie will live in the browser, whether it can only be sent over HTTPS, what domains it applies to, if it can be accessed using JavaScript, and more. Cookies are inherently more secure since they have these additional options, localStorage does not."*
Meeting expectations **(Score = 3)**	• Provides a clear and concise answer to 60-85% of the questions provided • Occasionally uses examples of certain technologies or concepts *Example Question- What is the difference between GET vs POST?* *Example answer : "GET and POST are both HTTP verbs that are used when making HTTP requests. GET can be used for fetching information, searching and sorting and POST can be used for sending data to the server to modify things. POST requests have side-effects and GET requests do not. GET requests send data in the query string and POST requests send data in the body of the request"*

	Second Example - "What are some differences between cookies and localStorage?" *Example answer - "Cookies and localStorage are both ways to store information in the browser. Typically you would store larger pieces of information in localStorage and smaller pieces of information in the browser. localStorage is set and read using JavaScript in the browser. Cookies are sent to the server on every single request from the same domain, localStorage does not automatically send data to the server on a request."*
Approaching expectations **(Score = 2)**	• Provides a clear and concise answer to 40-60% of the questions provided without any examples *Example answer: "GET and POST are for making HTTP requests using axios. GET is for getting information and POST is for sending information the server"* *Example answer- LocalStorage and cookies are both ways to store things. I've used them for authentication and for keeping track of logged in users.*
Not yet meeting expectations **(Score = 1)**	• Candidate is unable to answer at least 40% of questions

Feedback and conclusion - 8 minutes

Programming Rubric:

Grading Criteria	Not yet meeting expectations	Approaching expectations	Meeting expectations	Exceeding expectations
Project walkthrough	1	2	3	4
Technical knowledge	1	2	3	4

Total Score:

Fail (Score<=4)		Pass (Score>=5)	
Level 1: Not yet meeting expectations	Level 2: Approaching expectations	Level 3: Meeting expectations	Level 4: Exceeding expectations
1-2	3-4	5-6	7-8

Special Case: any candidate who scores "Not yet meeting expectations" (score = 1) in ANY of the sections, automatically fails the programming interview. This would indicate the student is unable to explain his/her technical solution design choices and has no clear structure or approach regarding how they have approached their solution.

Candidate Evaluation- 8 minutes

Your interviewer will be encouraged to share verbal or written feedback with you during the **final 10 minutes**. You will be given a score on a 4-point scale for internal consistency and calibration. The scoring rubric is listed below. A score of 3 represents a borderline PASS. Since companies err on the side of preventing false positives, it is very common for the average evaluation to be borderline; it's left up to the other interviewers on the panel to break the tie. A score of 4 represents a clear PASS. If the interviewer was hiring for a hypothetical company, they are satisfied and have confidence that the candidate understands the material necessary for a junior level role.

Coding Interview

Topic	Time Allocated
Coding Challenges	50 Minutes
Feedback and Conclusion	10 Minutes

The coding interview tests the candidate's ability to solve problems programmatically. This includes clarifying what is being asked, knowledge of algorithms and data structures, and coding skills such as correctness, testing, debugging, and code quality. Much of a Software Engineer's job is the "engineering" part, so this is a crucial (and often, by the candidate, underestimated) part of the hiring process. As a hiring manager, it's very common to have strong candidates fail to get hired due to poor coding performance.

Coding interviews are problem-oriented. The interviewer will prepare either two questions or a single question with one or more follow-ups.

Coding interviews will use the site https://coderpad.io. At the start of your interview, your mock interviewer will send you a unique link to the staging environment for the interview.

Candidates are allowed to use JavaScript or Python, and should state their preference. If a candidate wants to use a less common language such as Haskell, Go, or Rust, that's OK, too. The interviewer should be able to figure out what's going on in any mainstream language, even if they've never used it before, but a candidate must be able to explain what their code is doing. Testing and running inside the Coderpad environment helps confirm correctness. If you are using a language outside of the languages taught in this course, the interviewer may not know it, and it will be difficult to assess how idiomatic the candidate's code is.

General Flow

The mock interviewer will spend the first few minutes introducing themselves.

The mock interviewer will allocate **50 minutes** to coding tasks. **You should expect to complete at least Problem 1 and part of Problem 2. Do not worry if you do not get through**

everything, regardless of if you complete both problems the interviewer will be able to give you valuable feedback. Most candidates will need 10-20 minutes for the first problem and 30-40 minutes for the second problem.

At the beginning, you will be asked to join the Coderpad.io session. Your interviewer will begin by describing the first problem. It is very important to ask clarifying questions at this point: think about what kinds of inputs could create what kinds of issues, and ask questions about edge cases. You may not have enough information to fully solve the problem when it is given to you.

Once you understand the problem statement, walk your interviewer through the algorithm you want to use *in plain English*. Before you write code, discuss correctness, asymptotic time, and space complexity. You should be able to reason about BigO performance, tradeoffs such as time vs. space, and other properties of you chosen algorithms. You may mention general strategies into which your solutions fits — e.g. “divide and conquer.”

Now it’s time to start coding. Listen closely to your interviewer as you work towards your answer, they may have valuable insights about the solution you are working on. Be prepared to answer questions about the code you are writing. They may ask you about various the trade offs you are making as you work.

If you arrive at a solution with a suboptimal runtime or space complexity, you may be invited to write pseudocode for a more optimal solution.

After you complete the first problem, you will be given a second, more challenging problem. Make sure that you at least attempt this second problem. Only very strong candidates will be able to finish both problems in the time alloted, but if you do finish the interview ahead of time, use the last few minutes to clean up the code to production quality.

At the end of the interview, there will be some time devoted to testing and debugging. You will be driving the testing and debugging of code, but listen carefully to what your interviewer is saying, as they may have gentle hints for you.

Candidate Evaluation

Your interviewer will be encouraged to share verbal or written feedback with you during the **final 10 minutes**. You will be given a score on a 4-point scale for internal consistency and calibration. The scoring rubric is listed below. A score of 3 represents a borderline PASS. Since companies err on the side of preventing false positives, it is very common for the average evaluation to be borderline; it’s left up to the other interviewers on the panel to break the tie. A score of 4 represents a clear PASS. If the interviewer was hiring for a hypothetical

company, they are satisfied and have confidence that the candidate understands the material necessary for a junior level role.

Score	Observational Behaviours
Exceeding expectations **(Score = 4)**	• The candidate arrived at the correct solution to both parts independently • Time complexity and space complexity requirements are met • The code is clean and idiomatic *Example Question- Given an array of unique numbers and a number, return true if there exists a pair of numbers in the array that sum to the number given?* *Answer: The candidate solves this problem in O(N) time and O(N) space (using a set/map/object) or the candidate solves this problem in O(N log N) time and O(1) space (sorting and using pointers)* *Example Question 2 - Given two strings, return true if the second string is an anagram of the first string* *Answer - The candidate solves this problem in O(N) time using a data structure (map or object) and does not sort.*
Meeting expectations **(Score = 3)**	• The candidate arrived at the correct solution to the first problem with little help from the interviewer • Time complexity and space complexity requirements are met or pseudocode was written, or the candidate verbalized an approach to achieve the optimal complexities for the second problem • The code contains a few bugs which can be fixed with help from the interviewer *Example answer : The candidate solves this problem with the optimal time complexity with a hint from the interviewer (could you use a different data structure? Could we arrange the array in any particular way?)* *Example Question 2 - Given two strings, return true if the second string is an anagram of the first string*

	Answer - The candidate solves this problem in O(N) time with a good deal of help from the interviewer, or solves the problem using sorting (in N log N time)
Approaching expectations **(Score = 2)**	• Candidate can solve the first problem with significant help from the interviewer • Candidate can not arrive at the optimal solution for either problem even with significant help from the interviewer *Example answer: Candidate can only solve this problem in O(N^2) for both problems*
Not yet meeting expectations **(Score = 1)**	• Candidate is unable to solve any of the problems even with significant help from the interviewer

Feedback and conclusion - 10 minutes

Programming Rubric:

Grading Criteria	Not yet meeting expectations	Approaching expectations	Meeting expectations	Exceeding expectations
Coding Challenge	1	2	3	4

Total Score:

Fail (score<=2)		Pass (Score>=3)	
Level 1: Not yet meeting expectations	Level 2: Approaching expectations	Level 3: Meeting expectations	Level 4: Exceeding expectations
1	2	3	4

Whiteboarding

Topic	Time Allocated
Whiteboarding Challenge	50 Minutes
Feedback and Conclusion	10 Minutes

The whiteboarding interview invites a candidate to use their problem solving process & knowledge of programming to solve a programming problem on a whiteboard. The mock interviewer will present the prompt to the candidate and instruct them of how much time they have (50 min) and to treat it like a real world scenario. The prompt isn't meant to be a brainteaser or puzzle since the primary goal is to solve the problem correctly and as efficiently as possible. However, many times the problem is too difficult to solve in a short period of time, so interviewers are looking for a problem solving process and pseudocode instead. Candidates will be evaluated by gauging their knowledge of programming, problem solving, and ability to pseudocode and analyze algorithmic complexity.

The whiteboard challenges are open-ended enough so that it requires critical thinking, discussion, and collaboration. If a candidate is initially quick to solve the problem or provides a simple solution, the interviewer may encourage the candidate to be more rigorous and arrive at a more sophisticated solution. If a candidate answers Problem 1 easily and quickly, the interviewer will advance you to Problem 2 early. Otherwise, they will ask the candidate question about Problem 1. The first problem should take between 15-20 minutes and the second problem should take 25-30 minutes.

Whiteboarding Challenge - 50 minutes

The mock interviewer will open by introducing themselves for a few minutes. The interview will be done on video via Skype and the candidate will be instructed to screen share a Google Doc to conduct their whiteboarding interview. The interviewer should plan to spend 50 minutes on the task at hand. At the beginning, they will describe the problem to be solved and politely remind the candidate that they should think out loud throughout the whole exercise. Interviewers will answer any questions a candidate might have. Interviewers will encourage candidates to point out edge cases, explore alternate ideas, and test their assumptions.

Interviewers will keep candidates on track by pacing and reminding them how much time is left for the exercise. They will give candidates a heads up every 10 minutes or so to ensure the candidate is on track for coming to a resolution. Wrap up the exercise by asking "where could you improve this if you had more time?"

The **10 minutes** will be dedicated giving some verbal feedback.

Below is a process we encourage candidates to follow:

Process

- Listen carefully
- Repeat question back, rephrased
- Ask clarifying questions
- Write down requirements
- Write down test case(s)
- Stop and think!
- Pseudo-code
- Test your pseudocode
- Code!
- Test your code

Coming into the interview, you should be familiar with everything you learned in data structures and algorithms. You should also be familiar with the basic and intermediate functions of both Python and JavaScript. Remember to brush up on your array methods. Know how to use the built-in functions of scripts and arrays, and how to perform novel operations with strings and arrays. You will be asked more simple questions, like conceptual or practice questions you would find in the first two parts of an assessment. Then you will be asked more complex questions, like implementing merge sort or bubble sort. You might be asked to, for instance, see if any combination of letters can be arranged such that they form a palindrome. If the question is very difficult, break it down to the constituent parts and show how you would approach the problem.

Candidate Evaluation

Score	Observational Behaviours
Exceeding expectations **(Score = 4)**	• The candidate arrived at the correct solution to both parts independently • Time complexity and space complexity requirements are met • The code is clean and idiomatic *Example Question- Given an array of unique numbers and a number, return true if there exists a pair of numbers in the array that sum to the number given?* *Answer: The candidate solves this problem in O(N) time and O(N) space (using a set/map/object) or the candidate solves this problem in O(N log N) time and O(1) space (sorting and using pointers)*

Meeting expectations **(Score = 3)**	• The candidate arrived at the correct solution to both parts with some help from the interviewer • Time complexity and space complexity requirements are met or pseudocode was written, or the candidate verbalized an approach to achieve the optimal complexities • The code contains a few bugs which can be fixed with help from the interviewer *Example answer : The candidate solves this problem with the optimal time complexity with a hint from the interviewer (could you use a different data structure? Could we arrange the array in any particular way?)*
Approaching expectations **(Score = 2)**	• Candidate can not arrive at the optimal solution even with significant help from the interviewer *Example answer: Candidate can only solve this problem in O(N^2)*
Not yet meeting expectations **(Score = 1)**	• Candidate is unable to solve the problem regardless of time/space complexity

Feedback and conclusion - 10 minutes

Programming Rubric:

Grading Criteria	Not yet meeting expectations	Approaching expectations	Meeting expectations	Exceeding expectations
Whiteboarding Challenge	1	2	3	4

Total Score:

Fail (score<=2)		Pass (Score>=3)	
Level 1: Not yet meeting expectations	Level 2: Approaching expectations	Level 3: Meeting expectations	Level 4: Exceeding expectations
1	2	3	4

Candidate Evaluation

Your interviewer will be encouraged to share verbal or written feedback with you during the **final 10 minutes**. You will be given a score on a 4-point scale for internal consistency and calibration. The scoring rubric is listed below. A score of 3 represents a borderline PASS. Since companies err on the side of preventing false positives, it is very common for the average evaluation to be borderline; it's left up to the other interviewers on the panel to break the tie. A score of 4 represents a clear PASS. If the interviewer was hiring for a hypothetical company, they are satisfied and have confidence that the candidate understands the material necessary for a junior level role.