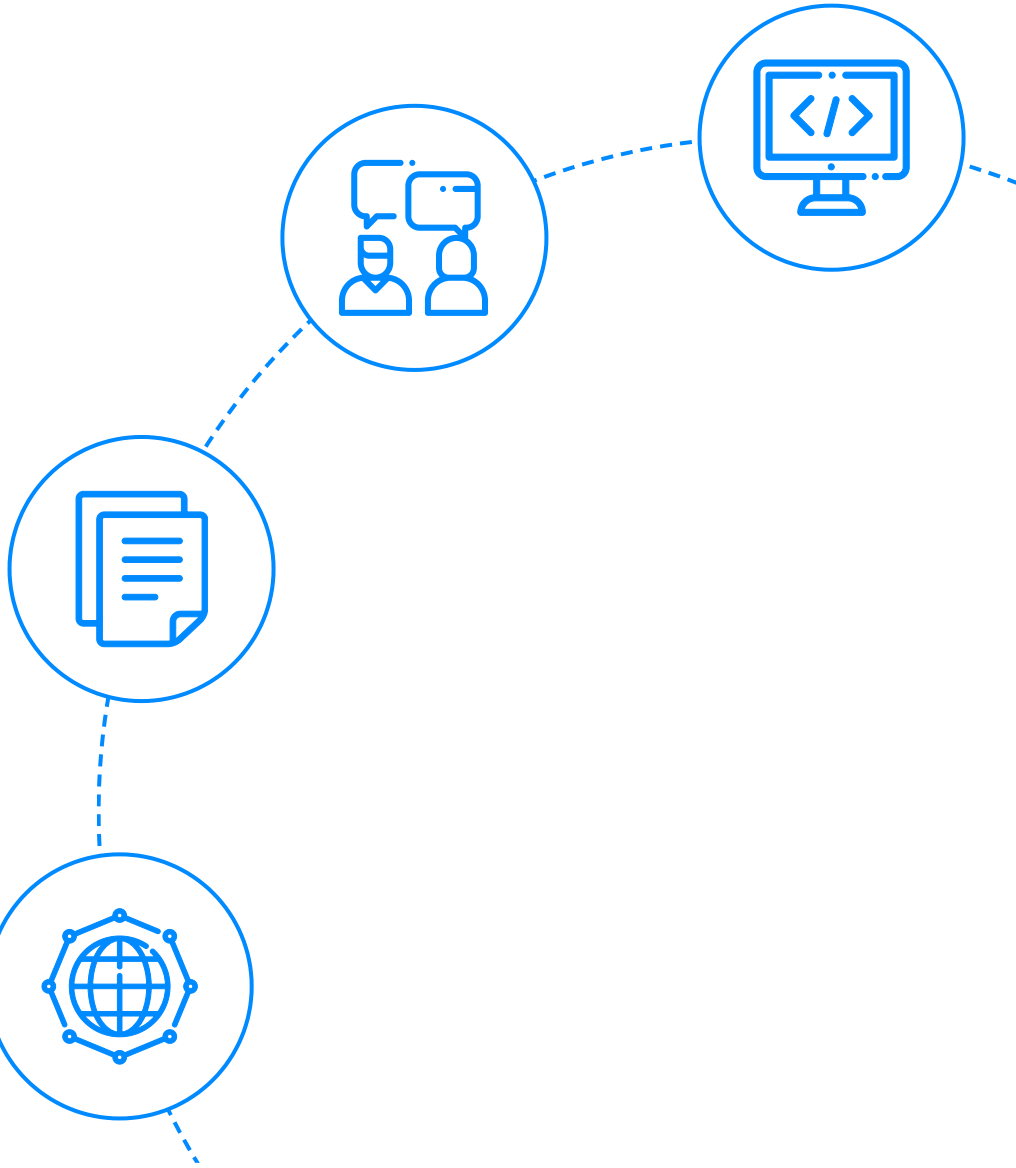
InterviewBit

SDET Interview Questions



To view the live version of the page, click here.

Contents

SDET Interview Questions for Freshers

SDET Interview Questions for Experienced

SDET Interview Questions for Experienced

(.....Continued)

19. Given the urgency with which a crucial hotfix must be deployed - What kind of testing technique would you use if you were in charge of the project?
20. What do you understand about fuzz testing? What are the types of bugs detected by fuzz testing?
21. What do you understand about a test script? Differentiate between test case and test script.
22. Differentiate between walkthrough and inspection.
23. What do you understand about white box testing and black box testing? Differentiate between them.

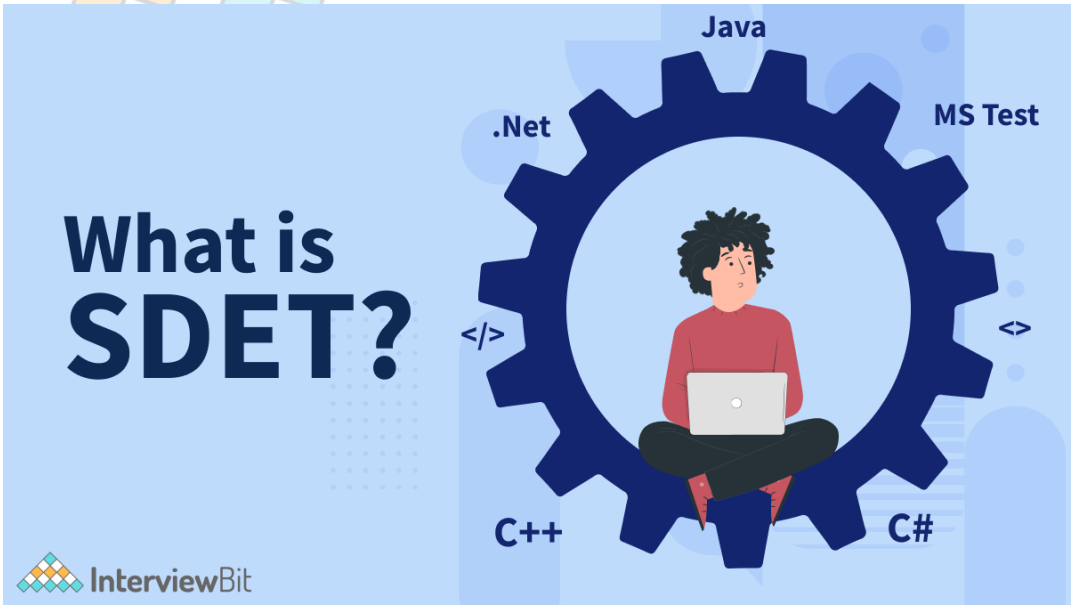
SDET Programming Interview Questions

24. Swap the values of two variables, x and y, without needing a third variable.
25. Write a program that reverses a given input number.
26. Write a programme to check whether the pairs and ordering of "", "", "(", ")", "[", "]" in the expression string expression are balanced or not.
27. Write a program to print the factorial of a given input number.
28. Determine whether there is a triplet in the array whose total equals the given value, given an array and a value. If a triplet of this type exists in the array, print it and return true. If not, return false.

Let's get Started

What is SDET?

In testing, an SDET (Software Development Engineer in Test) is an IT expert who can operate equally well in development and testing. SDETs are involved in every step of the software development and testing process. The knowledge of SDET professionals is entirely focused on software testing and development process testability, robustness, and performance. They can also participate as a contributor or reviewer in the development of production software designs. Currently, businesses are looking for a professional who can assist with software development. At the same time, he should be in charge of evaluating the software that has been produced. That is why hiring SDET is beneficial to them because they can work on high-performance code or testing framework creation.



Microsoft was the first to do so, but other companies are now taking notice, and they're seeking someone who is an expert in SDET to help them with the whole development of their product, as well as the testing design that must be carried out for that specific development. The company can use the same resource for two critical jobs that will always be profitable.

In this article, we have covered the most frequently asked interview questions for the profile of Software Development Engineer in Test (SDET). For a Software Development Engineer in Test (SDET), it is essential to be thorough with the principles of **software testing** as well.

SDET Interview Questions for Freshers

1. Differentiate between Software Development Engineer in Test (SDET) and Manual Tester.

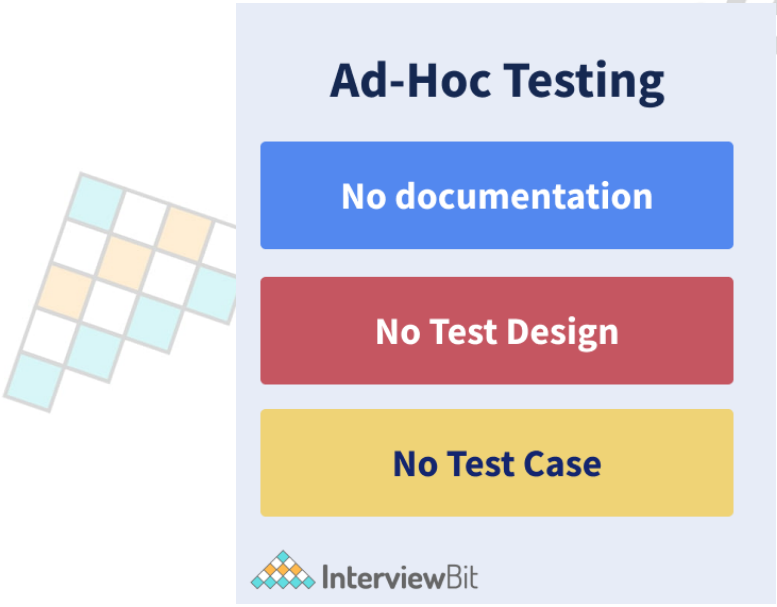
Tester: A tester is someone who performs software testing in order to find flaws. The tester additionally examines several features of the software. The tester is unaware of the software development process. A tester examines the software to see whether it contains any faults or flaws.

The following table lists the differences between an SDET and a Manual Tester:

Software Development Engineer in Test (SDET)	Manual Tester
SDET refers to a tester who is also a coder.	A tester tests software or systems after they have been developed.
SDET is well-versed in the areas of design, implementation, and testing.	A tester is unaware of the software's design and implementation.
SDET also examines the software's performance.	The tester is only responsible for testing duties.
SDET is well-versed in software requirements and other related topics.	A tester has a limited understanding of software requirements.
SDET is involved at every stage of the software development life cycle.	In the software development life cycle, testers play a less role and have fewer obligations.
SDET needs to be well versed in coding since they can be required to do both automated and manual testing.	Testers need not be well versed in coding since they are only required to do manual testing

2. What do you understand about ad-hoc testing?

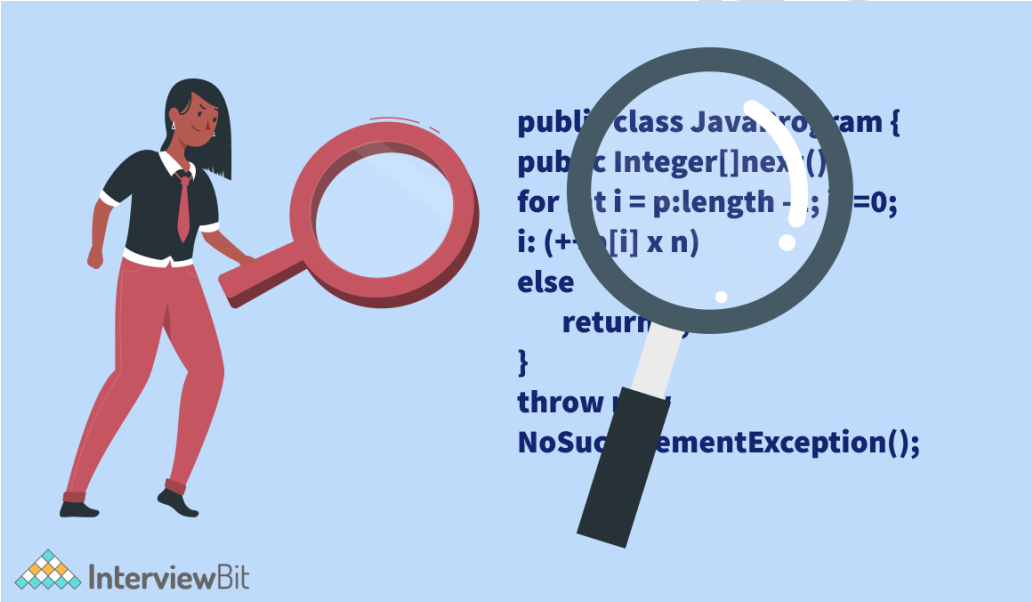
Ad hoc testing is a type of unstructured or informal software testing that seeks to interrupt the testing process in order to uncover potential defects or errors as soon as feasible. Ad hoc testing is a type of testing that is done at random and is usually an unplanned activity that does not use any documentation or test design methodologies to construct test cases.



Ad-hoc testing is done on any portion of the application at random and does not follow any standardized testing procedures. The primary goal of this testing is to detect problems through random inspection. Error Guessing, a software testing approach, can be used to perform ad hoc testing. People with adequate familiarity with the system can "predict" the most likely source of errors, which is known as error guessing. This testing does not necessitate any paperwork, planning, or procedure. Because this testing seeks to detect faults through a random technique, defects will not be mapped to test cases if there is no documentation. This means that reproducing errors can be difficult at times because there are no test processes or requirements associated with them.

3. What do you understand about code inspection in the context of software testing? What are its advantages?

Code inspection is a sort of static testing that involves inspecting software code and looking for flaws. It simplifies the initial error detection procedure, lowering the defect multiplication ratio and avoiding subsequent stage error detection. This code inspection is actually part of the application evaluation procedure.



Following are the **key steps** involved in code inspection :

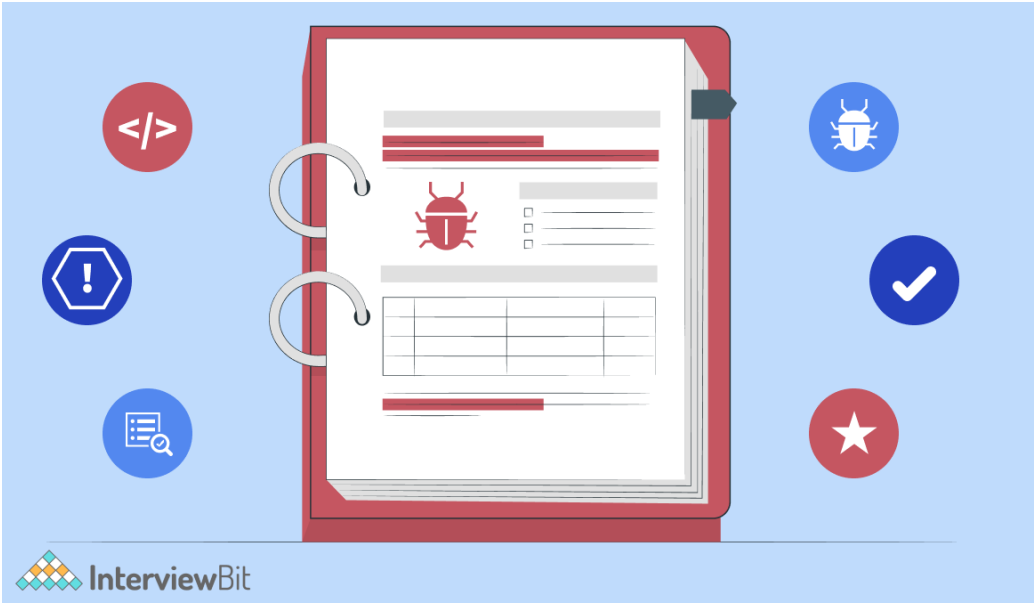
- An Inspection team's primary members are the Moderator, Reader, Recorder, and Author.
- The inspection team receives related documents, prepares the inspection meeting, and coordinates with the inspection team members.
- If the inspection team is unfamiliar with the project, the author gives them an overview of the project and its code.
- Following that, each inspection team conducts a code inspection using inspection checklists.
- Conduct a meeting with all team members after the code inspection is completed to discuss the code that was inspected.

Following are the **advantages** of Code Inspection :

- Code Inspection enhances the overall quality of the product.
- It finds bugs and flaws in software code.
- In any event, it marks any process improvement.
- It finds and removes functional defects in a timely and effective manner.
- It aids in the correction of prior flaws.

4. What is a bug report in the context of software testing?

A bug report is a detailed report which explains what is incorrect and needs to be fixed in software or on a website. The report includes a request and/or details for how to address each issue, as well as a list of causes or noticed faults to point out exactly what is perceived as wrong. Bug reports are a technique to inform developers about parts of their code that aren't behaving as expected or designed, allowing them to see which parts of their software need to be improved. This can be a difficult effort for the developer, and without enough information, it is nearly impossible. Fortunately, testers may make this process considerably easier by producing high-quality bug reports that include all of the information a developer might need to locate the problem.



5. What are the elements of a bug report?

Following are the elements of a bug report :

- **TITLE -** A good title is simple, concise, and provides a description of the bug to the developer. It should include the bug's categorization, the app component where the bug happened (e.g. Cart, UI, etc.), and the activity or conditions in which the bug occurred. A clear title makes it easier for the developer to find the report and distinguishes duplicate reports, making problem triaging much easier.
- **SEVERITY AND PRIORITY -** The severity of an issue is determined by its severity. The severity levels and definitions vary amongst programme developers, and even more so between developers, testers, and end-users who are unaware of these distinctions. The standard categorization is as follows:
    - **Critical/Blocker:** This category is reserved for faults that render the application useless or result in significant data loss.
    - **High**: When a bug affects a major feature and there is no workaround or the remedy that is provided is extremely complicated.
    - **Medium:** The bug affects a minor or significant feature, but there is a simple enough fix to avoid major discomfort.
    - **Low:** This is for defects that have a modest impact on the user experience, such as minor visual bugs.
- **DESCRIPTION -**This is a concise summary of the bug, including how and when it occurred. This section should contain additional information than the title, such as the frequency with which the bug happens if it is an intermittent error and the situations that appear to trigger it. It contains information about how the bug is impacting the application.
- **ENVIRONMENT -** Apps can behave in a variety of ways depending on their surroundings. This section should contain all of the information about the app's environment setup and settings.
- **REPRO STEPS -** This should include the bare essentials for reproducing the bug. The steps should ideally be short, easy, and accessible to anybody. The goal is for the developer to be able to reproduce the error on their end in order to figure out what's wrong. A bug report without repro steps is useless and wastes time and effort that could be better spent resolving more complete reports; make sure to convey this to your testers and in a way that your end-users understand.
- **ACTUAL RESULT -** This is what the tester or user saw as a result or output.
- **EXPECTED RESULT -** This is the anticipated or planned consequence or output.
- **ATTACHMENTS -** Attachments can assist the developer find the problem faster; a screenshot of the problem can explain a lot, especially when the problem is visual. Logs and other incredibly useful attachments can at the very least put the

6. What are the do’s and don'ts for a good bug report?

Following are the **do’s** for a good bug report :

- When you're finished, read your report. Make sure it's clear, concise, and simple to understand.
- Don't allow any opportunity for ambiguity by being as clear as possible.
- Do test the problem a few times to see if there are any superfluous procedures.
- Include any workarounds or additional procedures you've discovered that cause the problem to behave differently in your report.
- Check to see if the bug has previously been reported. If it has, please leave a comment on the bug with your information.
- Do reply to requests for further information from developers.

Following are the **don’ts** for a good bug report :

- DO NOT submit a report that contains more than one bug. When there are numerous bugs in the report, keeping track of their progress and dependencies becomes difficult.
- DO NOT be judgmental or accusatory. Bugs are unavoidable, yet they aren't always simple to fix.
- DO NOT try to figure out what's causing the bug. Stick to the facts to avoid sending the developer on a wild goose hunt.
- Anything that isn't a bug should be posted. Developers appreciate hearing from you, but sending information to the wrong channel will just block their workflow and cause delays.

7. What are the roles and responsibilities of a Software Development Engineer in Test (SDET)?

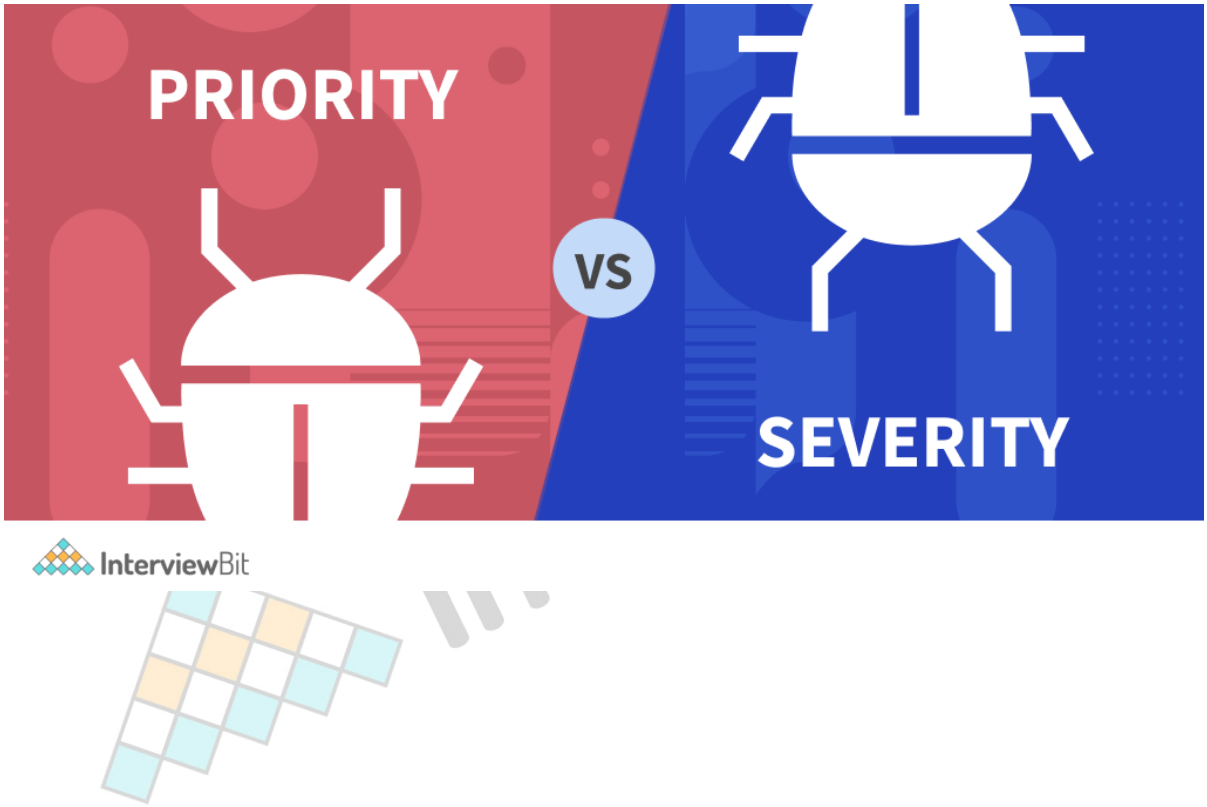
Following are the roles and responsibilities of a Software Development Engineer in Test (SDET) :

- SDETs should be able to automate tests and set up frameworks on a variety of platforms, including web, mobile, and desktop.
- Investigate customer issues that have been referred to you by the technical support staff.
- Create and manage bug reports, as well as interact with the rest of the team.
- Ability to create various test scenarios and acceptability tests.
- SDET is responsible for handling technical discussions with Partners in order to gain a better understanding of the client's systems or APIs.
- SDET also works with deployment teams to resolve any system-level difficulties.
- SDETs should also be able to create, maintain, and run test automation frameworks.

8. What do you understand about severity and priority in the context of software testing? Differentiate between them.

- **Severity** - Severity in testing refers to how much of an impact it has on the computer program under test. A higher severity rating indicates that the bug/defect has a greater impact on system functionality. The severity level of a bug or defect is usually determined by a Quality Assurance engineer.
- **Priority** - The order in which a fault should be repaired is referred to as a priority. The higher the priority, the faster the problem should be fixed. Flaws that render the software system unworkable are prioritized over defects that affect only a tiny portion of the software's functionality.

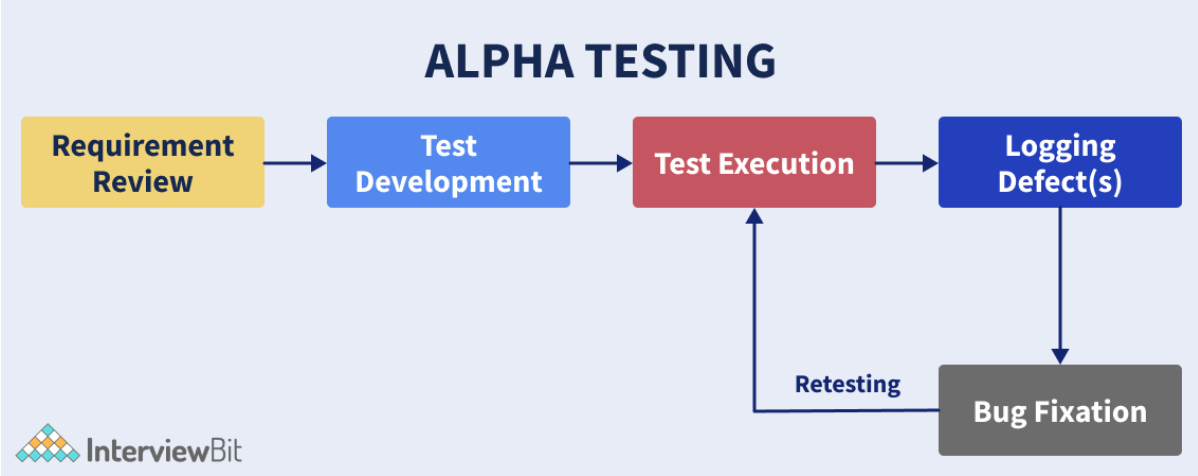
The following table lists the differences between priority and severity -



Priority	Severity
The sequence in which a developer should fix a bug is determined by priority.	The severity of a defect is defined as the impact it has on the product's operation.
Priority is divided into three categories : • Low • Medium • High The bugs are usually assigned priority labels like P0, P1, P2 and so on, where P0 denotes the bug with the highest priority.	There are five types of severity : • Critical • Major • Moderate • Minor • Cosmetic
Priority is linked to the scheduling of bugs in order to resolve them.	Severity is linked to the functionality or standards of the application.
Priority denotes how urgently the fault should be corrected.	Severity denotes the severity of the defect's impact on the product's functionality.
In consultation with the manager/client, the priority of defects is determined.	The defect's severity level is determined by the QA engineer.
Its worth is subjective and might fluctuate over time based on the project's circumstances.	Its worth is objective and unlikely to fluctuate.

9. What do you understand about alpha testing? What are its objectives?

Alpha testing is a type of software testing that is used to find issues before a product is released to real users or the general public. One type of user acceptability testing is alpha testing. It is referred to as alpha testing since it is done early in the software development process, near the ending. Homestead software developers or quality assurance staff frequently undertake alpha testing. It's the final level of testing before the software is released into the real world.



Following are the **objectives of alpha testing**:

- The goal of alpha testing is to improve the software product by identifying flaws that were missed in prior tests.
- The goal of alpha testing is to improve the software product by identifying and addressing flaws that were missed during prior tests.
- The goal of alpha testing is to bring customers into the development process as early as possible.
- Alpha testing is used to gain a better understanding of the software's reliability during the early phases of development.

10. What do you understand about beta testing? What are the different types of beta testing?

Genuine users of the software application undertake beta testing in a real environment. One type of User Acceptance Testing is beta testing. A small number of end-users of the product are given a beta version of the program in order to receive input on the product quality. Beta testing reduces the chances of a product failing and improves the product's quality by allowing customers to validate it.

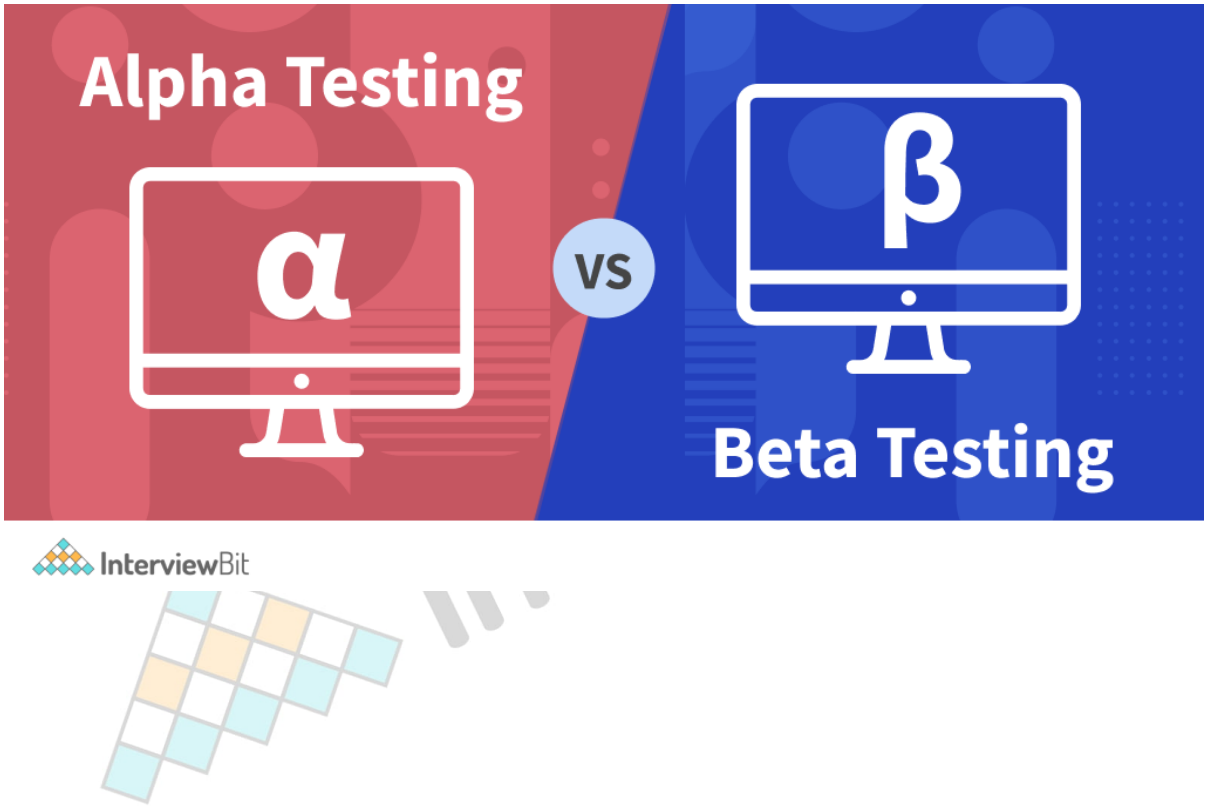
Following are the different **types of beta testing** :

- **Traditional Beta testing:** Traditional Beta testing is distributing the product to the target market and collecting all relevant data. This information can be used to improve the product.
- **Public Beta Testing:** The product is made available to the general public via web channels, and data can be gathered from anyone. Product improvements can be made based on customer input. Microsoft, for example, undertook the largest of all Beta Tests for its operating system Windows 8 prior to its official release.
- **Technical Beta Testing:** A product is delivered to a group of employees of a company and feedback/data is collected from the employees.
- **Focused Beta Testing:** A software product is distributed to the public for the purpose of gathering input on specific program features. For instance, the software's most important features.
- **Post-release Beta Testing:** After a software product is launched to the market, data is collected in order to improve the product for future releases.

SDET Interview Questions for Experienced

11. Differentiate between Alpha testing and Beta testing.

The following table lists the differences between alpha testing and beta testing:



Alpha testing	Beta testing
Both white box and black box testing are used in alpha testing.	Black box testing is used in beta testing.
Alpha testing is frequently done by testers who are inside employees of the company.	Clients who are not employees of the company undertake beta testing.
Alpha testing takes place on the developer's premises.	Beta testing is done on the product's end-users.
Alpha testing does not include any reliability or security testing.	During beta testing, reliability, security, and robustness are examined.
Before moving on to beta testing, alpha testing verifies that the product is of high quality.	Beta testing focuses on the product's quality as well as gathering user feedback and ensuring that the product is ready for real-world use.
Alpha testing necessitates the use of a lab or a testing environment.	Beta testing does not necessitate the use of a testing setting or laboratory.
Alpha testing could necessitate a lengthy execution cycle.	Only a small amount of time is required for beta testing.

12. Mention some of the software testing tools used in the industry and their key features.

Following are some of the **software testing tools** used in the industry :

- **TestRail** - TestRail is a web-based test case management system that is scalable and flexible. Our cloud-based/SaaS solution may be set up in minutes, or you can install your own server on TestRail.

- **Testpad** - Testpad is a more straightforward and accessible manual testing tool that emphasizes pragmatism over method. It employs checklist-inspired test plans that may be modified to a broad range of approaches, including exploratory testing, the manual side of Agile, syntax-highlighted BDD, and even traditional test case management, rather than handling cases one at a time.

- **Xray** - Xray is a full-featured tool that exists inside Jira and works flawlessly with it. Its mission is to assist businesses in improving the quality of their products through efficient and effective testing.

- **Practitest** - PractiTest is a complete test management solution. It provides comprehensive visibility into the testing process and a better, broader understanding of testing outcomes by serving as a common meeting ground for all QA stakeholders.

- **SpiraTest** - SpiraTest is a cutting-edge Test Management solution for both large and small teams. Spiratest allows you to handle requirements, plans, tests, issues, tasks, and code in a unified environment, fully embracing the agile method of working. SpiraTest is ready to use right out of the box and adapts to your needs, methodology, workflows, and toolchain.

- **TestMonitor** - Every firm can benefit from TestMonitor's end-to-end test management capabilities. It is a straightforward and intuitive way to test. TestMonitor has you covered whether you're adopting corporate software, need QA, produce a quality app, or just need a helping hand with your test project.

13. What do you understand about performance testing and load testing? Differentiate between them.

- **Performance Testing:** Performance testing is a sort of software testing that guarantees that software programmes function as expected under certain conditions. It is a method of determining system performance in terms of sensitivity, reactivity, and stability when subjected to a specific workload. The practice of examining a product's quality and capability is known as performance testing. It's a means of determining how well a system performs in terms of speed, reliability, and stability under various loads. Perf Testing is another name for performance testing.
- **Load Testing:** Load testing is a type of performance testing that assesses a system's, software product's, or software application's performance under realistic load situations. Load testing determines how a programme behaves when several users use it at the same time. It is the system's responsiveness as assessed under various load conditions. Load testing is done under both normal and excessive load circumstances.

The following table lists the differences between Performance Testing and Load Testing :

Performance Testing	Load Testing
The process of determining a system's performance, which includes speed and reliability under changing loads, is known as performance testing.	The practice of determining how a system behaves when several people access it at the same time is known as load testing.
The load on which the system is evaluated is normal in terms of performance.	In load testing, the maximum load is used.
It examines the system's performance under regular conditions.	It examines the system's performance under heavy load.
The limit of load in performance testing is both below and above the break threshold.	The limit of load in load testing refers to the point at which a break occurs.
It verifies that the system's performance is satisfactory.	It determines the system's or software application's working capacity.
During performance testing, speed, scalability, stability, and reliability are all examined.	During load testing, only the system's long-term viability is examined.
Tools for performance testing are less expensive.	Load testing equipment is expensive.

14. Explain some expert opinions on how a tester can determine whether a product is ready to be used in a live environment.

Because this is such a crucial decision, it has never been taken by a single individual or by junior guys. This choice is not made solely by the developer and tester; senior management is involved on a regular basis. Management tests primarily ensure that product delivery is bug-free by validating the following:

- Validating the bug reports that the tester has submitted. How the bug was fixed and whether or not the tester retested it.
- Validating all of the test cases written by the tester for that specific functionality, as well as the documentation and confirmation received from the tester.
- Run automated test cases to ensure that new features do not interfere with existing features.
- Validating test coverage report, which confirms that all of the developing component's test cases have been written.

15. What do you understand about Risk based testing?

Risk-based testing (RBT) is a method of software testing that is based on risk likelihood. It entails analyzing the risk based on software complexity, business criticality, frequency of use, and probable Defect areas, among other factors. Risk-based testing prioritizes testing of software programme aspects and functions that are more important and likely to have flaws.

Risk is the occurrence of an unknown event that has a positive or negative impact on a project's measured success criteria. It could be something that happened in the past, something that is happening now, or something that will happen in the future. These unforeseen events might have an impact on a project's cost, business, technical, and quality goals.

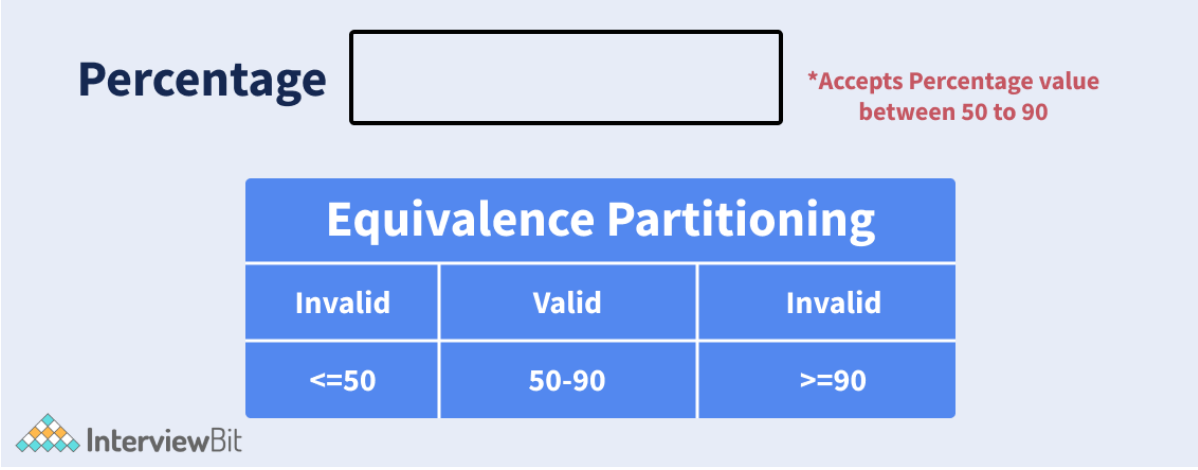
Risks can be positive or negative. Positive risks are referred to as opportunities, and they aid in the long-term viability of a corporation. Investing in a new project, changing corporate processes, and developing new products are just a few examples.

Negative risks are also known as threats, and strategies to reduce or eliminate them are necessary for project success.

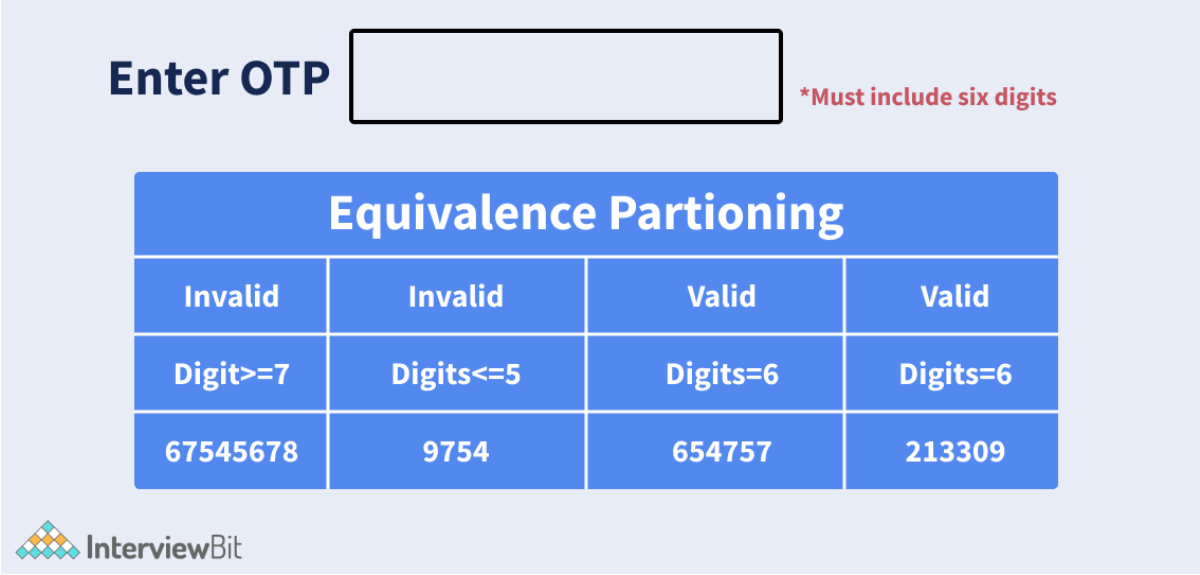
16. What is Equivalence Partitioning, and how does it work? Use an example to demonstrate your point.

Equivalence Class Partitioning (ECP) is another name for the Equivalence Partitioning Method. It is a software testing technique, often known as black-box testing, that splits the input domain into data classes from which test cases can be constructed. An ideal test case identifies a type of error that may necessitate the execution of a large number of arbitrary test cases before a general error is detected. Equivalence classes are evaluated for given input conditions in equivalence partitioning. When any input is given, the type of input condition is examined, and the Equivalence class defines or explains a collection of valid or invalid states for this input condition.

- **Example 1** - Let's take a look at a typical college admissions procedure. There is a college that admits students depending on their grade point average. Consider a percentage field that will only accept percentages between 50 and 90%; anything higher or lower will result in the program redirecting the visitor to an error page. If the user enters a percentage that is less than 50% or greater than 90%, the equivalence partitioning technique will display an invalid percentage. The equivalence partitioning method will show a valid percentage if the percentage entered is between 50 and 90%.



- **Example 2** - Consider the following software application as an example. A software application has a function that accepts only a certain number of numbers, not even larger or fewer than that number. Consider an OTP number with only six digits; anything more or less than six numbers will be rejected, and the application will route the customer or user to an error page. If the user's password is fewer than or equal to six characters, the equivalence partitioning method will display an invalid OTP. The equivalence partitioning technique will display a valid OTP if the password given is precisely six characters.



17. How would you go about creating an Automation Strategy for a product that doesn't have any automation tests?

These types of questions are open-ended, so you may take the conversation in any direction you desire. You can also highlight your strong talents, knowledge, and technological areas.

You can, for example, use instances of the Automation Strategy you used while constructing a product in a previous capacity to respond to these types of inquiries.

For example, you could say things like,

- Because the product necessitated starting automation from the ground up, you had plenty of time to consider and design an appropriate automation framework, opting for a language/technology that the majority of people were familiar with in order to avoid introducing a new tool and instead leverage existing knowledge.
- You began by automating the most basic functional scenarios, which were referred to as P1 scenarios (without which no release could go through).
- You also considered using automated test tools like JMETER, LoadRunner, and others to test the system's performance and scalability.
- You considered automating the application's security features as outlined in the OWASP Security guidelines.
- For early feedback and other reasons, you included automated tests into the build workflow.

18. Would you forego thorough testing in order to release a product quickly?

These questions usually ask the interviewer to grasp your thinking as a leader, as well as what you would compromise on and whether you would be prepared to produce a flawed product in exchange for less time.
Answers to these questions should be supported by the candidate's actual experiences.

For example, you may say that in the past, you had to make a decision to release a hotfix, but it couldn't be tested since the integration environment was unavailable. So you rolled it out in stages, starting with a small proportion and then monitoring logs/events before launching the complete rollout, and so on.

19. Given the urgency with which a crucial hotfix must be deployed - What kind of testing technique would you use if you were in charge of the project?

In this case, the interviewer is most interested in learning more about you.

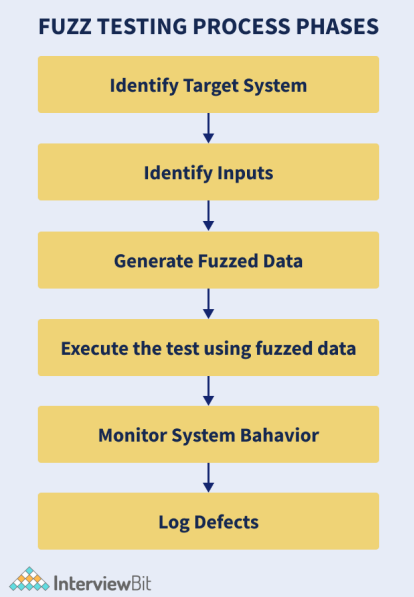
- What types of test strategies can you conceive of and how do you plan to implement them?
- What kind of coverage would you provide in the event of a hotfix?
- How would you test the hotfix after it's been deployed? etc.

If you can relate to the problem, you can use real-life circumstances to answer such questions. You should also explain that you would not be willing to deliver any code to production without proper testing.
When it comes to essential fixes, you should always collaborate with the developer to figure out what areas they might affect and set up a non-production environment to test the change. It's also worth noting that you'll continue to monitor the fix (using monitoring tools, dashboards, logs, and so on) after it's been deployed to look for any anomalous behaviour in the production environment and ensure that the fix hasn't had any negative consequences.

20. What do you understand about fuzz testing? What are the types of bugs detected by fuzz testing?

Fuzz Testing is a software testing technique that utilizes erroneous, unexpected, or random data as input and then looks for exceptions like crashes and memory leaks. It's a type of automated testing that's used to define system testing techniques that use a randomized or dispersed approach. During fuzz testing, a system or software program may have a variety of data input problems or glitches.

Following are the different phases of Fuzz Testing:



- **Identify the Target System:** The system or software application that will be tested is identified. The target system is the name given to that system. The testing team determines the target system.
- **Identify Inputs:** Once the target system is determined, random inputs are generated for testing purposes. The system or software application is tested using these random test scenarios as inputs.
- **Generate Fuzzed Data:** After receiving unexpected and invalid inputs, these invalid and unexpected inputs are turned to fuzzed data. Fuzzed data is essentially random data that has been transformed into fuzzy logic.
- **Use fuzzed data to run the test:** The fuzzed data testing process is now being used. Basically, the code of the programme or software is executed in this portion by providing random input, i.e. fuzzed data.
- **Monitor System Behavior:** After the system or software application has completed its execution, check for any crashes or other anomalies, such as potential memory leaks. The random input is used to test the system's behaviour.
- **Log Flaws:** In the final phase, defects are detected and rectified in order to produce a higher-quality system or software program.

Following are the different **types of bugs detected by fuzz testing**:

- **Failures in assertions and memory leaks** - This practice is often employed in large applications where defects compromise memory safety, which is a serious flaw.
- **Invalid data** - Fuzzers are used in fuzz testing to generate faulty input that is used to test error-handling methods, which is critical for software that does not have control over its input. Simple fuzzing is a technique for automating negative testing.
- **Correctness bugs** - Some forms of "correctness" flaws can also be detected using fuzzing. For example, a corrupted database, inadequate search results, and so on.

21. What do you understand about a test script? Differentiate between test case and test script.

Test scripts are a line-by-line description of the system transactions that must be done in order to validate the application or system under test. Each step should be listed in the test script, along with the intended outcomes. This automation script enables software testers to thoroughly test each stage on a variety of devices. The actual items to be executed, as well as the expected results, must be included in the test script.

The following table lists the differences between test case and test script :

Test Case	Test Script
A test case is a detailed technique for testing an application.	A test script is a set of instructions for autonomously testing an application.
In a manual testing environment, Test Cases are employed.	In the automation testing environment, Test Script is employed.
It's done by hand.	It is carried out in accordance with the scripting format.
Test ID, test data, test technique, actual and predicted outcomes, and so on are all included in the test case template.	To create a script in Test Script, we can utilise a variety of commands.

22. Differentiate between walkthrough and inspection.

Walkthrough - A walkthrough is a technique for doing a quick group or individual review. In a walkthrough, the author describes and explains his work product to his peers or supervisor in an informal gathering to receive comments. The legitimacy of the suggested work product solution is checked here. It is less expensive to make adjustments while the design is still on paper rather than during conversion. A walkthrough is a form of quality assurance that is done in a static manner. Walkthroughs are casual gatherings with a purpose.

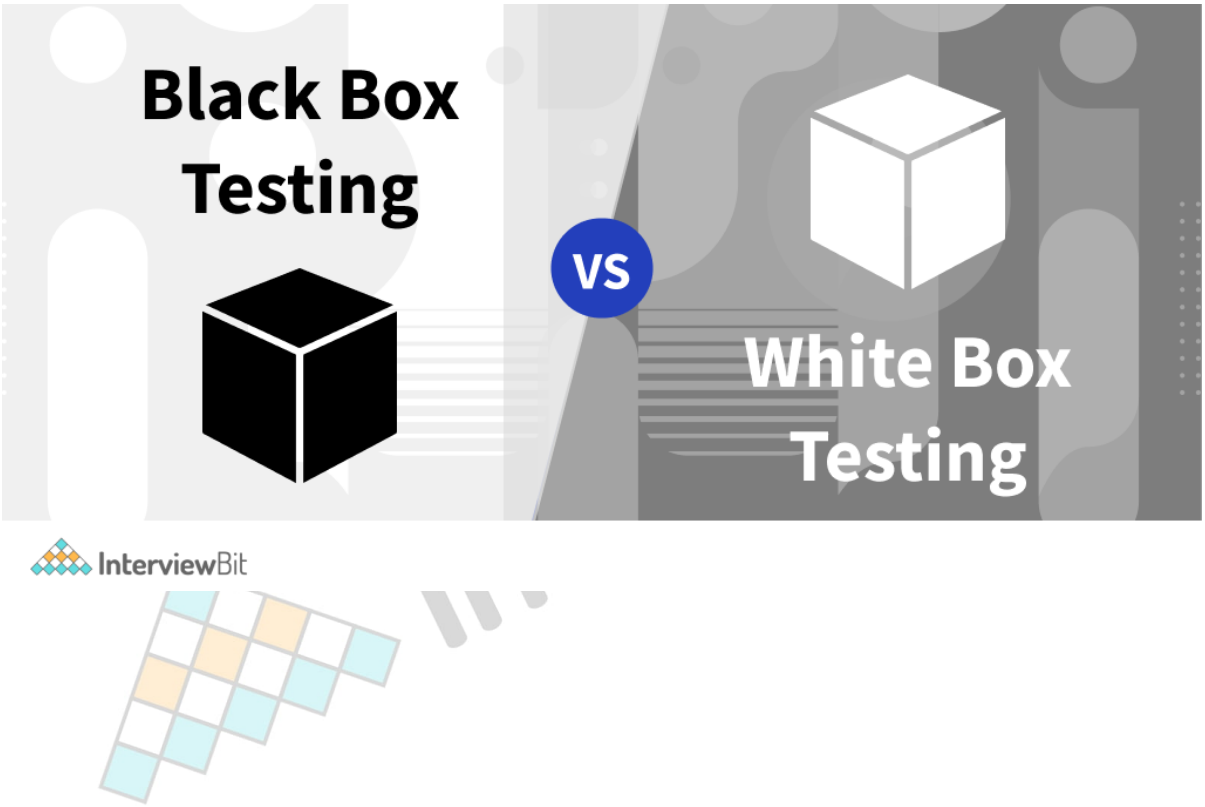
The following table lists the differences between walkthrough and inspection -

Walkthrough	Inspection
It is informal in nature.	It is formal in nature.
The developer starts it.	The project team starts it.
The developer of the product takes the lead throughout the walkthrough.	The inspection is conducted by a group of people from several departments. The tour is usually attended by members of the same project team.
The walkthrough does not employ a checklist.	A checklist is used to identify flaws.
Overview, little or no preparation, little or no preparation examination (real walkthrough meeting), rework, and follow up are all part of the walkthrough process.	Overview, preparation, inspection, rework, and follow-up are all part of the inspection process.
In the steps, there is no set protocol.	Each phase has a formalized protocol.
Because there is no specific checklist to evaluate the programme, the walkthrough takes less time.	An inspection takes longer since the checklist items are checked off one by one.
It is generally unplanned in nature.	Scheduled meeting with fixed duties allocated to all participants.
	The responsibility of the

23. What do you understand about white box testing and black box testing? Differentiate between them.

- **Black Box Testing:** The customer's statement of requirements is the most common source of black-box testing. It's a different kind of manual testing. It's a software testing technique that looks at the software's functionality without knowing anything about its internal structure or coding. It does not necessitate any software programming knowledge. All test cases are created with the input and output of a certain function in mind. The test engineer examines the programme against the specifications, detects any faults or errors, and returns it to the development team.
- **White Box Testing:** The term "clear box," "white box," or "transparent box" refers to the capacity to see into the inner workings of software through its outside shell. It is carried out by developers, after which the software is handed to the testing team for black-box testing. The fundamental goal of white-box testing is to examine the infrastructure of an application. It is carried out at a lower level, as unit testing and integration testing are included. It necessitates programming skills because it primarily focuses on a program's or software's code structure, routes, conditions, and branches. The main purpose of white-box testing is to concentrate on the flow of inputs and outputs via the software while also ensuring its security.

The following table lists the differences between black box and white box testing:



Black Box Testing	White Box Testing
It's a software testing technique that looks at how the software works without knowing anything about its core structure or coding.	In white-box testing, the tester is aware of the software's internal structure.
Functional testing, data-driven testing, and closed-box testing are all terms used to describe Black Box Testing.	It is also known as structural testing, clear box testing, code-based testing, and transparent testing.
There is minimal programming knowledge necessary in black-box testing.	There is a demand for programming skills in white-box testing.
It's not the best tool for testing algorithms.	It's ideal for algorithm testing and comes highly recommended.
It is carried out at the higher stages of testing, such as system and acceptability testing.	It is carried out at the unit and integration testing stages of testing.
It is primarily carried out by software testers.	Developers are the ones who do it the most.
It takes less time to complete. The amount of time spent on black-box testing is determined by the availability of functional specifications.	It takes a longer time. Because of the large code, creating test cases takes a long time.

SDET Programming Interview Questions

24. Swap the values of two variables, x and y, without needing a third variable.

- **Approach 1:**

The goal is to find a sum in one of the two numbers provided. The total and subtraction from the sum can then be used to swap the integers.

Code:

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
   int a = 1, b = 2;

   cout << "Before Swapping : a = " << a << " and b = " << b << "\n";
   a = a + b; // storing the sum of a and b in a
   b = a - b; // storing the value of the original a in b
   a = a - b; // storing the value of the original b in a
   cout << "After Swapping : a = " << a << " and b = " << b << "\n";
}
```

Output :

```
Before Swapping : a = 1 and b = 2
After Swapping : a = 2 and b = 1
```

Explanation :
In the above code, we first stored the sum of both the numbers in the first variable. Then, we store the original value of the first variable in the second variable by subtracting the second variable from the sum. Similarly we change the value for the second variable as well. Thus, we swapped the two numbers without using a third variable.

- **Approach 2 :**

To swap two variables, use the bitwise XOR operator. When two integers x and y are XORed, the result is a number with all bits set to 1 wherever the bits of x and y differ. For instance, the XOR of 10 (in Binary 1010) and 5 (in Binary 0101) is 1111, while the XOR of 7 (0111) and 5 (0101) is 1111. (0010). We can then xor the resultant XORed with the other number to swap the values. Considering the above example, when we xor 1111 with 0101 we get 1010.

Code :

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
   int a = 1, b = 2;

   cout << "Before Swapping : a = " << a << " and b = " << b << "\n";
   a = a ^ b; // storing the xor of a and b in a
   b = a ^ b; // storing the value of the original a in b
   a = a ^ b; // storing the value of the original b in a
   cout << "After Swapping : a = " << a << " and b = " << b << "\n";
}
```

Output :

```
Before Swapping : a = 1 and b = 2
After Swapping : a = 2 and b = 1
```

Explanation :

In the above code, we first stored the xor of both the numbers in the first variable. Then, we store the original value of the first variable in the second variable by XORing the second variable with the sum. Similarly, we change the value for the second variable as well. Thus, we swapped the two numbers without using a third variable.

25. Write a program that reverses a given input number.

Example :

Input:

```
1234
```

Output:

```
4321
```

Approach:

We follow a **greedy approach**. We start extracting each digit from the original number from the end and keep on adding it at the end of the new number, thereby reversing the given number.

Code:

```
#include <bits/stdc++.h>
using namespace std;
//function to find the reverse of the given input
int reverseNumber(int original_number)
{
   int new_number = 0;
   while (original_number > 0) {
       new_number = new_number * 10 + original_number % 10; // extracting the last digi
       original_number = original_number / 10;
   }
   return new_number;
}

int main()
{
   int original_number = 1234;
   cout << "The original number is " << original_number << "\n";
   cout << "The reversed number is " << reverseNumber(original_number) << "\n";
   return 0;
}
```

Output :

```
The original number is 1234
The reversed number is 4321
```

Explanation :

In the above code, the function reverseNumber takes the input of an integer number and returns the reverse of the number. We extract each digit from the end and append it to the new number.

26. Write a programme to check whether the pairs and ordering of "", "", "(", ")", "[", "]" in the expression string expression are balanced or not.

By balanced string, it is meant that every opening parenthesis should have its corresponding closing braces and there must not be redundant closing braces.

Example :

Input :

```
“[()]{}{[()()]()}”
```

Output :

```
Balanced
```

Input:

```
[[
```

Output:

```
Not Balanced
```

Approach:

We create a character stack and now look through the expression string. If the current character is a beginning bracket ('(', '', or '['), stack it. If the current character is a closing bracket (') or '' or ']'), pop from the stack; if the popped character is the matching starting bracket, all is well; otherwise, the brackets are unbalanced. If some starting brackets remain in the stack after traversal, the stack is said to be "unbalanced."

Code:

```
#include <bits/stdc++.h>
using namespace std;

// function to check if the given expression is balanced or not
bool checkBalancedExpression(string str)
{
   stack<char> char_stack;
   char top_element;

   // Traversing the Expression
   for (int i = 0; i < str.length(); i++)
   {
       if (str[i] == '(' || str[i] == '[' || str[i] == '{')
       {
           // We push the character in the stack
           char_stack.push(str[i]);
           continue;
       }

       // We check if the stack is empty. If it is empty, then it is an unbalanced expr
       if (char_stack.empty())
           return false;

       switch (str[i]) {
       case ')':


           top_element = char_stack.top();
           char_stack.pop();
           if (top_element == '{' || top_element == '[')
               return false;
           break;

       case '}':

           top_element = char_stack.top();
           char_stack.pop();
           if (top_element == '(' || top_element == '[')
               return false;
           break;

       case ']':

           top_element = char_stack.top();
           char_stack.pop();
           if (top_element == '(' || top_element == '{')
               return false;
           break;
       }
   }

   // The stack should be empty at this point for a balanced expression
   return (char_stack.empty());
}

int main()
{
```

Output :

```
Balanced
```

Explanation:

In the above code, we create a function checkBalancedExpression which takes the input of an expression string and checks whether the string is balanced or not. We start traversing the expression. Whenever we encounter an opening bracket, we push it in the stack. In the case of a closing bracket, we first check if the stack is empty or not. If it is empty, we directly return false otherwise, we check if the top element has the corresponding opening bracket or not.

27. Write a program to print the factorial of a given input number.

Example:

Input:

```
5
```

Output:

```
120
```

In practically all interviews, factorial is one of the most frequently asked questions (including developer interviews) Developer interviews place a greater emphasis on programming concepts such as dynamic programming, recursion, and so on, whereas Software Development Engineer in Test interviews place a greater emphasis on handling edge scenarios such as max values, min values, negative values, and so on, and approach/efficiency becomes secondary.

The following recursive formula can be used to calculate factorial.

```
n! = n * (n-1)!
n! = 1 if n = 0 or n = 1
```

Code:

```
#include <bits/stdc++.h>
using namespace std;

// function to find the factorial of given number
int findFactorial(unsigned int n)
{
   if (n == 0)
       return 1;
   return n * factorial(n - 1);
}

int main()
{
   int number = 5;
   if(number < 0)
   {
       cout << "Negative numbers do not have factorial" << "\n";
       return 0;
   }
   cout << "Factorial of " << number << " is " << findFactorial(number) << “\n”;
   return 0;
}
```

Output:

```
Factorial of 5 is 120
```

Explanation:

In the above code, the function findFactorial finds the factorial of a given input number. Before making the function call, we check if the given number is negative. Negative numbers do not have a factorial and so we must display appropriate messages for it.

28. Determine whether there is a triplet in the array whose total equals the given value, given an array and a value. If a triplet of this type exists in the array, print it and return true. If not, return false.

Example:

Input:

```
arr = {12, 3, 4, 1, 6, 9}, sum = 24;
```

Output:

```
12, 3, 9
```

Explanation:

The array contains a triplet (12, 3 and 9) whose sum is 24.

Input:

```
arr = {1, 2, 3, 4, 5}, sum = 9
```

Output:

```
5, 3, 1
```

Explanation:

There is a triplet (5, 3 and 1) present in the array whose sum is 9.

Approach:

The algorithm's performance can be increased by sorting the array. The two-pointer technique is used in this effective method. Fix the first element of the triplet by traversing the array. Find if there is a pair whose total equals x – array[i] using the Two Pointers technique. Because the two pointers approach takes linear time, it is preferable to a nested loop.

Code:

```
#include <bits/stdc++.h>
using namespace std;

// returns true if there is triplet with sum equal
// to 'sum' present in arr[]. Also, prints the triplet
bool findTriplet(int arr[], int arr_size, int sum)
{
   int low, high;

   //sorting the entire array
   sort(arr, arr + arr_size);

   //Fixing the first element one by one and finding the other two elements
   for (int i = 0; i < arr_size - 2; i++) {

       // Start two index variables from opposite corners of the array and move them to
       low = i + 1; // index of the first element in the remaining elements

       high = arr_size - 1; // index of the last element
       while (low < high) {
           if (arr[i] + arr[low] + arr[high] == sum) {
               printf("The triplet is %d, %d, %d", arr[i],
                      arr[low], arr[high]);
               return true;
           }
           else if (arr[i] + arr[low] + arr[high] < sum)
               low++;
           else // arr[i] + arr[low] + arr[high] > sum
               high--;
       }
   }

   // If we reach here, then no triplet was found
   return false;
}

int main()
{
   int arr[] = { 12, 3, 4, 1, 6, 9 };
   int sum = 24;
   int arr_size = sizeof(arr) / sizeof(arr[0]);

   findTriplet(arr, arr_size, sum);

   return 0;
}
```

Output:

```
The triplet is 12, 3, 9
```

Explanation:

In the above code, the function findTriplet takes the input of an array, the size of the array and the sum required. First of all, we sort the array given. We traverse the array elements and fix the first element of the triplet in each iteration. We use a two-pointer approach and find the remaining two elements of the triplet, if it is possible with the given array.

Useful Resources:

- QA Interview Questions
- Automation Testing Interview Questions
- API Testing Interview Questions
- Selenium Interview Questions
- Software Development Engineer: Career Guide

Links to More Interview Questions

C Interview Questions	Php Interview Questions	C Sharp Interview Questions
Web Api Interview Questions	Hibernate Interview Questions	Node Js Interview Questions
Cpp Interview Questions	Oops Interview Questions	Devops Interview Questions
Machine Learning Interview Questions	Docker Interview Questions	Mysql Interview Questions
Css Interview Questions	Laravel Interview Questions	Asp Net Interview Questions
Django Interview Questions	Dot Net Interview Questions	Kubernetes Interview Questions
Operating System Interview Questions	React Native Interview Questions	Aws Interview Questions
Git Interview Questions	Java 8 Interview Questions	Mongodb Interview Questions
Dbms Interview Questions	Spring Boot Interview Questions	Power Bi Interview Questions
Pl Sql Interview Questions	Tableau Interview Questions	Linux Interview Questions
Ansible Interview Questions	Java Interview Questions	Jenkins Interview Questions