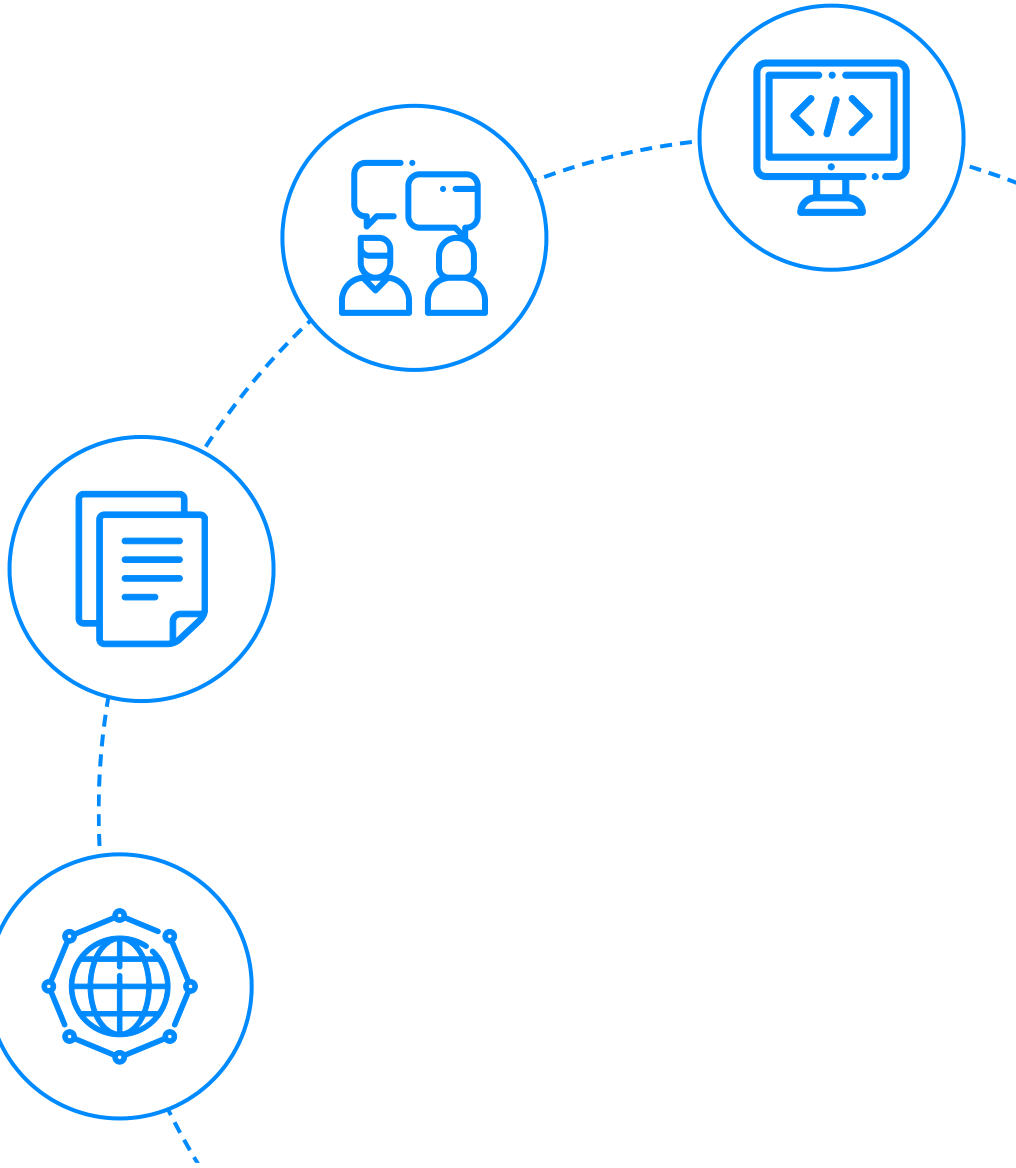
InterviewBit

Software Testing Interview Questions



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

Contents

Software Testing Interview Questions For Freshers

Software Testing Interview Questions For Experienced

21. How much testing is sufficient? Or, is it possible to do exhaustive testing of the software?
22. Why developers shouldn't test the software they wrote?
23. What qualities a software tester should have?
24. What is a bug report?
25. What are some important testing metrics?
26. What is Test-Driven-Development?
27. What is Selenium? What are its benefits?
28. What is boundary value analysis?
29. What are the differences between manual and automated testing?
30. What is a user story?
31. What is an API?
32. What are the different HTTP status codes that a server can return?
33. What is API testing?
34. What is a test environment?
35. What is test coverage?
36. What is meant by browser automation?
37. What is A/B testing?
38. What is meant by Code Coverage?
39. List some of the popular testing tools/frameworks, providing a brief description of each.
40. What are the different types of severity you can assign to a bug?

Conclusion

41. Conclusion

Let's get Started

Software testing is an activity conducted in the software development life-cycle to verify that the software is accurate and works according to the requirements. Testing plays an integral part in any software development project.

In its essence, software testing aims to answer the question: **How does one ensure that the software does what it is supposed to do and doesn't do what it is not supposed to do?** The primary goal behind software testing is to get enough confidence that the software under testing produces the correct output for a given input.

An important thing to keep in mind when learning about software testing is that testing does not improve software quality by itself. Or that a high amount of testing doesn't mean that the software is high quality. Testing is an indicator of quality, providing crucial feedback to the developers who created the software to take necessary action to fix the problems found in testing.

This article provides the frequently asked interview questions by the interviewer for a Software Tester or Quality Assurance (QA) position. It's divided into two sections based on the skill set of the applicant. The fresher's section includes software testing questions you might get asked for a junior-level position if you are recently graduated from college.

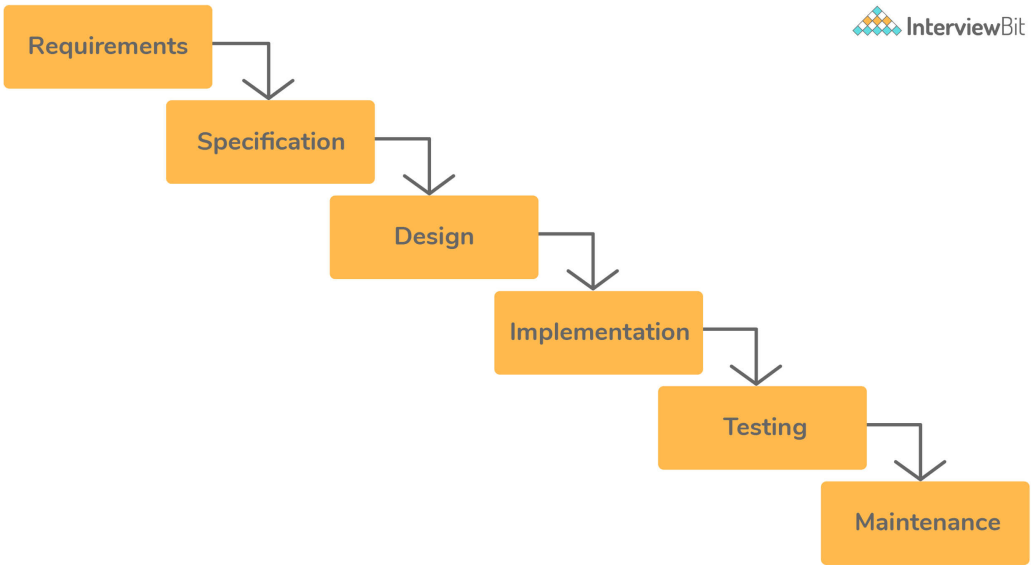
The experienced section assumes basic familiarity with the testing process and explores the advanced topics in testing. This section is suitable for someone with a few years of experience as a tester. In the end, multiple-choice questions are provided to test your understanding of testing.

Software Testing Interview Questions For Freshers

1. Explain the role of testing in software development?

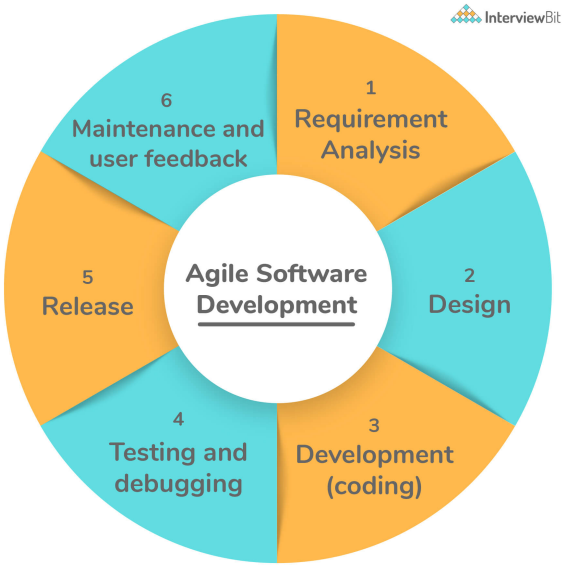
Software testing comes into play at different times in different software development methodologies. There are two main methodologies in software development, namely Waterfall and Agile.

In a traditional waterfall software development model, requirements are gathered first. Then a specification document is created based on the document, which drives the design and development of the software. Finally, the testers conduct the testing at the end of the software development life cycle once the complete software system is built.



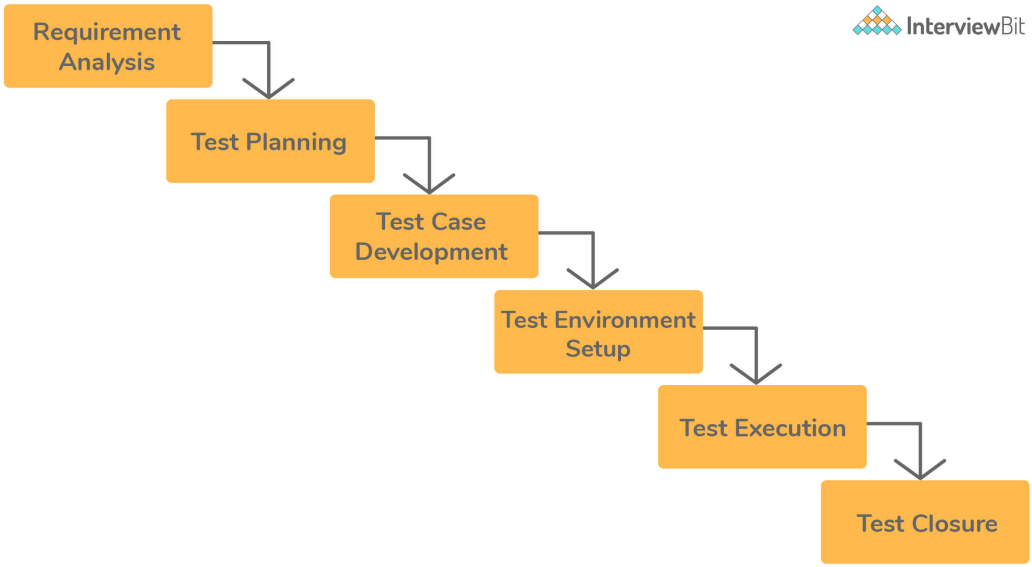
Waterfall Software Development Model

An agile software development model works in small iterations. You test the software in parallel as it is getting built. The developers build a small functionality according to the requirements. The testers test it and get customer feedback, which drives future development.



2. What is the software testing life cycle?

Similar to software development, testing has its life cycle. During the testing, a tester goes through the following activities.



1. **Understand the requirements:** Before testing software or a feature, the tester must first understand what it is supposed to do. If they don't know how the software is supposed to work, they can't test it effectively.
2. **Test Planning and Case Development:** Once the tester has a clear understanding of the requirements, they can create a test plan. It includes the scope of testing, i.e., part of software under test and objectives for testing. Various activities are involved in planning the test, such as creating documentation, estimating the time and efforts involved, deciding the tools and platforms, and the individuals who will be conducting the tests.
3. **Prepare a test environment:** The development happens in a development environment, i.e., on a developer's computer that might not represent the actual environment that the software will run in production. A tester prepares an environment with the test data that mimics the end user's environment. It assists with realistic testing of the software.
4. **Generate the test data:** Though it is impossible to do exhaustive testing of the software, the tester tries to use realistic test data to give them the confidence that the software will survive the real world if it passes the tests.
5. **Test Execution:** Once the tester has a complete understanding of the software and has a test environment set up with the test data, they execute the test. Here, execution means that the tester runs the software or the feature under test and verifies the output with the expected outcome.
6. **Test Closure:** At the end of the test execution, there can be two possible outcomes. First, the tester finds a bug in the part of the software under test. In this case, they create a test record/bug report. Second, the software works as expected. Both these events indicate the end of the test cycle.

3. What is functional testing?

Functional testing is a form of black-box testing. As the name suggests, it focuses on the software's functional requirements rather than its internal implementation. A functional requirement refers to required behavior in the system, in terms of its input and output.

It validates the software against the functional requirements or the specification, ignoring the non-functional attributes such as performance, usability, and reliability.

Functional testing aims to answer the following questions, in particular:

- Does the software fulfill its functional requirements?
- Does it solve its intended users' problems?

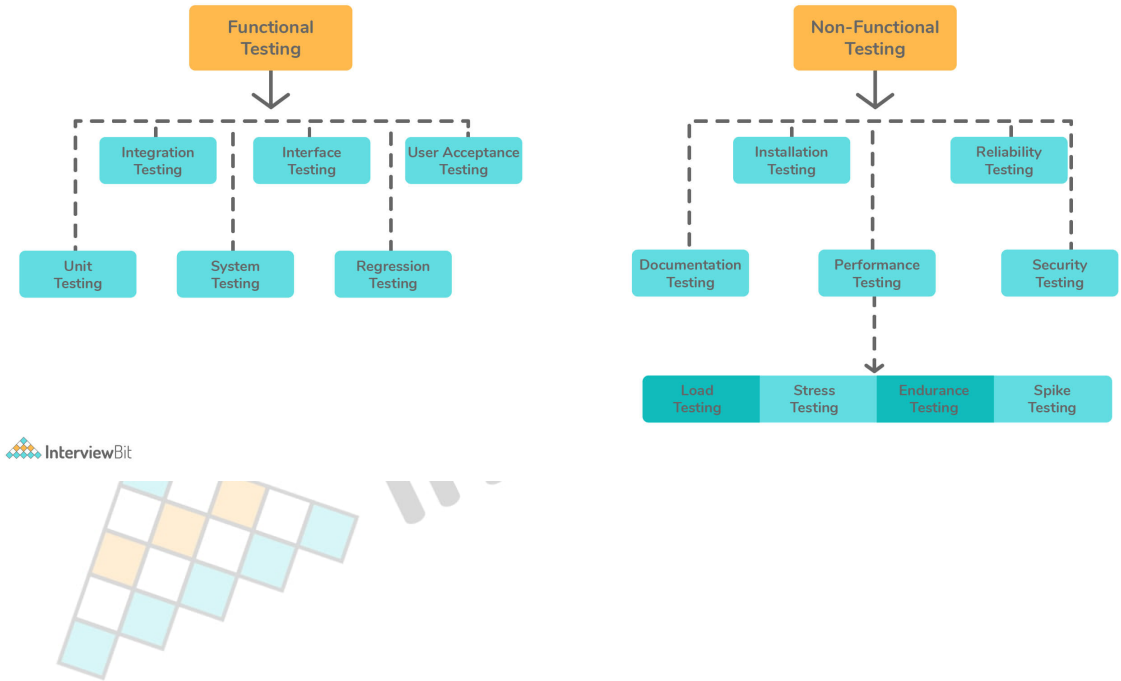
4. What is non-functional testing?

Non-functional testing tests the system's non-functional requirements, which refer to an attribute or quality of the system explicitly requested by the client. These include performance, security, scalability, and usability.

Non-functional testing comes after functional testing. It tests the general characteristics unrelated to the functional requirements of the software. Non-functional testing ensures that the software is secure, scalable, high-performance, and won't crash under heavy load.

5. What are the different types of testing?

You can test the software in many different ways. Some types of testing are conducted by software developers and some by specialized quality assurance staff. Here are a few different kinds of software testing, along with a brief description of each.



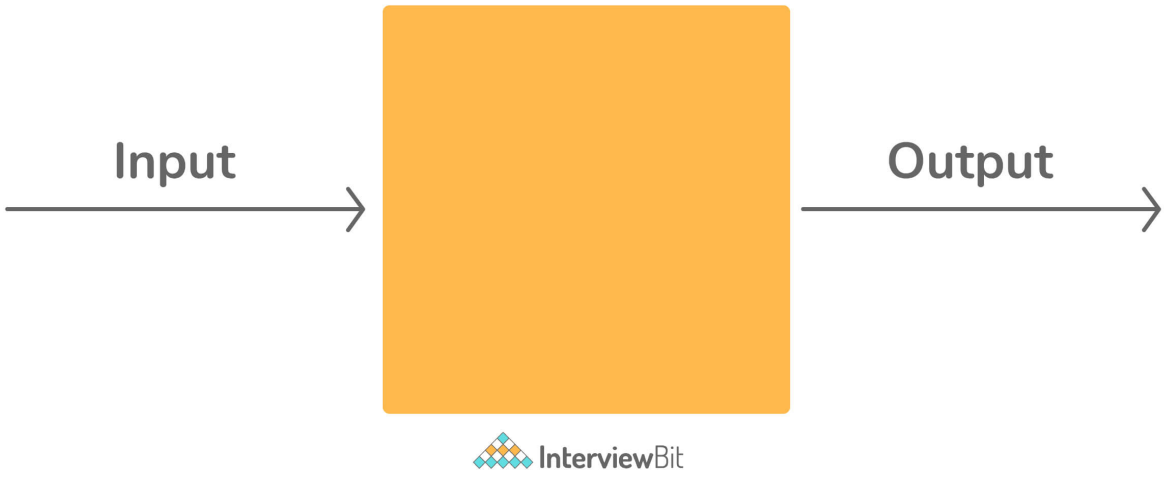
Type	Description
Unit Testing	A programmatic test that tests the internal working of a unit of code, such as a method or a function.
Integration Testing	Ensures that multiple components of systems work as expected when they are combined to produce a result.
Regression Testing	Ensures that existing features/functionality that used to work are not broken due to new code changes.
System Testing	Complete end-to-end testing is done on the complete software to make sure the whole system works as expected.
Smoke Testing	A quick test performed to ensure that the software works at the most basic level and doesn't crash when it's started. Its name originates from the hardware testing where you just plug the device and see if smoke comes out.
Performance Testing	Ensures that the software performs according to the user's expectations by checking the response time and throughput under specific load and environment.
User-Acceptance Testing	Ensures the software meets the requirements of the clients or users. This is typically the last step before the software is live, i.e. it goes to production.

6. What is black-box testing?

In black-box testing, the tester views the software as a black box, ignoring all the internal structure and behavior. Their only concern is the input provided to the system and the generated output. Black-box testing verifies the program's behavior against the specified requirements.

During black-box testing, the test conditions are created based upon the software's functionality but are unaware of how the software works internally. The software is tested from the end user's perspective and gives a broader picture of the whole system.

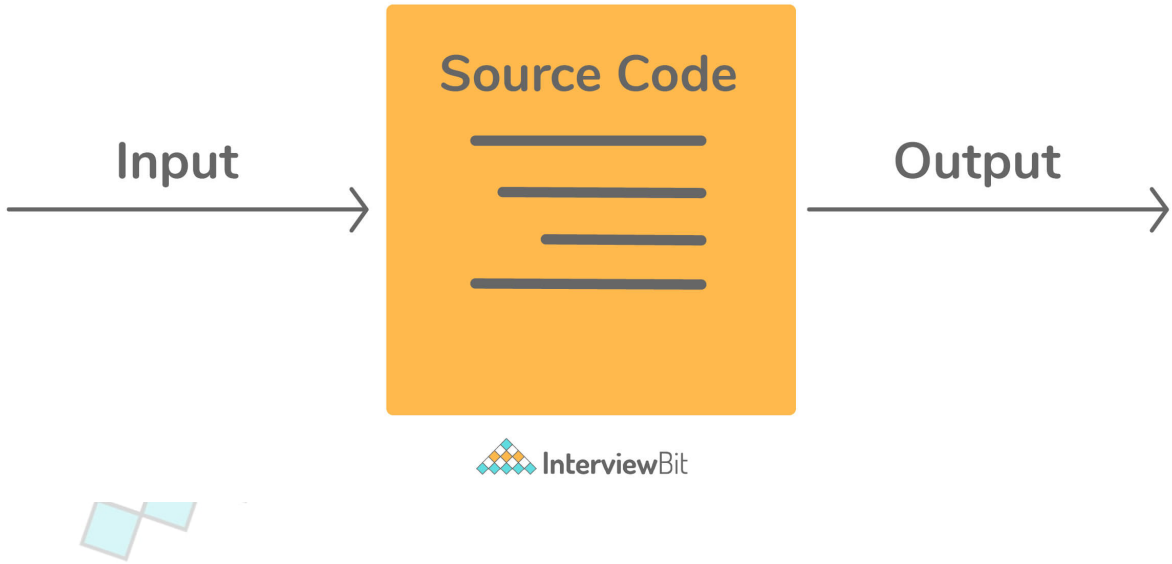
Given that the users are only concerned with whether the software works according to their needs and don't care how it works, black-box testing helps test software usability and anticipate how the customer will use the product.



7. What is white-box testing?

White-box testing is an alternative strategy to black-box testing, in which a tester views the system as a transparent box. They are allowed to observe the internal implementation of the system, which guides the test. Typically, the software developers perform the white-box testing during the development phase.

In white-box testing, we assume that the tester has some programming knowledge. They try to test each possible branch a program could take in a running system. Knowing what's inside the box, i.e., taking a look at the source code and the implementation details, it's possible to test the system more thoroughly.



8. What is manual testing?

In manual testing, a tester manually verifies the functionality of the software. The tester has a comprehensive list of all the test cases they should test, along with the test data. They go through each case, one by one. They launch the software as an end-user would, enter the input, and manually verify the output.

It may seem that manual testing is inefficient when compared to automated testing. It is slow, not repeatable in a consistent manner, and prone to human misjudgment.

However, manual testing allows the tester to realistically test the software, using actual user data in a natural user environment, subject to similar external conditions. Only a human, not a computer, can evaluate the usability and accessibility of the application and how it looks and feels to the end-user. It also gives a broader perspective of the system. Finally, some test scenarios just can't be automated and need to be manually tested.

You should always test the software manually before trying to automate the testing.

9. What is cross-browser testing?

All web applications run in browsers such as Google Chrome, Mozilla Firefox, Internet Explorer, Safari, etc. Though they all work primarily the same in implementing the web standards, there are subtle differences in all of them. When building the software, it’s not always possible for the software developer to meticulously test the feature on multiple browsers, noticing the subtle inconsistencies.

In cross-browser testing, a software tester launches the web application in all the supported browsers and tries to test the same functionality on all of them. They note any unexpected behavior in a browser that doesn't work as expected or looks different; note the behavior and the browser name and version in the test report. This helps the programmer to fix the behavior in all the browsers where it doesn't work as intended.

10. What is automated testing?

As the name suggests, automated testing, which is also called test automation, is the programmatic execution of the tests. The tester uses an automation tool or software like Selenium to write code that performs the following tasks.

1. Automatically run the software.
2. Feed the input data to the system.
3. Examine the output with the expected outcome.
4. Fail the test if the results don't match. Otherwise, pass the test.

Once a test is automated, you can run it as often as you want, to check if any new code has broken it. It enables you to spend your time on other high-value tests, such as exploratory testing that help find bugs that an automated test would miss.

Automated testing is beneficial for repetitive testing with inputs that don't change frequently. Humans get tired and bored from conducting the same tests repeatedly and seeing the same results. It's easy to make mistakes when you are testing a feature for the twentieth time. Software is much better at doing repetitive tasks without getting tired or making mistakes than a human operator would.

11. What is alpha testing?

Before you ship the software to the customers, the internal testing team performs alpha testing. Alpha testing is part of the user acceptance testing. Its goal is to identify bugs before the customers start using the software.

12. What is beta testing?

Once you ship the software to the customers after alpha testing, the software's actual users perform the beta testing in a real production environment. It is one of the final components of user acceptance testing. Beta testing is helpful to get feedback from real people using your software in real environments.

13. What is regression testing?

The dictionary definition of regression is the act of going back to a previous place or state. In software, regression implies that a feature that used to work suddenly stopped working after a developer added a new code or functionality to the software.

Regression problems are pervasive in the software industry, as new features are getting added all the time. Developers don't build these features in isolation, separate from the existing code. Instead, the new code interacts with the legacy code and modifies it in various ways, introducing side effects, whether intended or not.

As a result, there is always a chance that introducing new changes may negatively impact a working feature. It's important to keep in mind that even a small change has the potential to cause regression.

Regression testing helps ensure that the new code or modifications to the existing code don't break the present behavior. It allows the tester to verify that the new code plays well with the legacy code.

14. What is exploratory testing?

Imagine a tourist in a foreign city. There are two ways in which they can explore the city.

- Follow a map, itinerary, or a list of places they should visit
- Explore randomly, following the streets as they lead them to new places

With the first approach, the tourist follows a predetermined plan and executes it. Though they may visit famous spots, they might miss out on hidden, more exciting places in the city. With the second approach, the tourist wanders around the city and might encounter strange and exotic places that the itinerary would have missed.

Both approaches have their pros and cons.

A tester is similar to a tourist when they are testing software. They can follow a strict set of test cases and test the software according to them, with the provided inputs and outputs, or they can explore the software.

When a tester doesn't use the test scripts or a predefined test plan and randomly tests the software, it is called exploratory testing. As the name suggests, the tester is exploring the software as an end-user would. It's a form of black-box testing.

In exploratory testing, the tester interacts with the software in whatever manner they want and follows the software's instructions to navigate various paths and functionality. They don't have a strict plan at hand.

Exploratory testing primarily focuses on behavioral testing. It is effective for getting familiar with new software features. It also provides a high-level overview of the system that helps evaluate and quickly learn the software.

Though it seems random, exploratory testing can be powerful in an experienced and skilled tester's hands. As it's performed without any preconceived notions of what software should and shouldn't do, it allows greater flexibility to the tester to discover hidden paths and problems along those paths.

15. What is end-to-end testing?

End to End testing is the process of testing a software system from start to finish. The tester tests the software just like an end-user would. For example, to test a desktop software, the tester would install the software as the user would, open it, use the application as intended, and verify the behavior. Same for a web application.

There is an important difference between end-to-end testing vs. other forms of testing that are more isolated, such as unit testing. In end-to-end testing, the software is tested along with all its dependencies and integrations, such as databases, networks, file systems, and other external services.

16. What is unit testing?

Unit testing is the process of testing a single unit of code in an isolated manner. The unit of code can be a method, a class, or a module. Unit testing aims to focus on the smallest building blocks of code to get confidence to combine them later to produce fully functioning software.

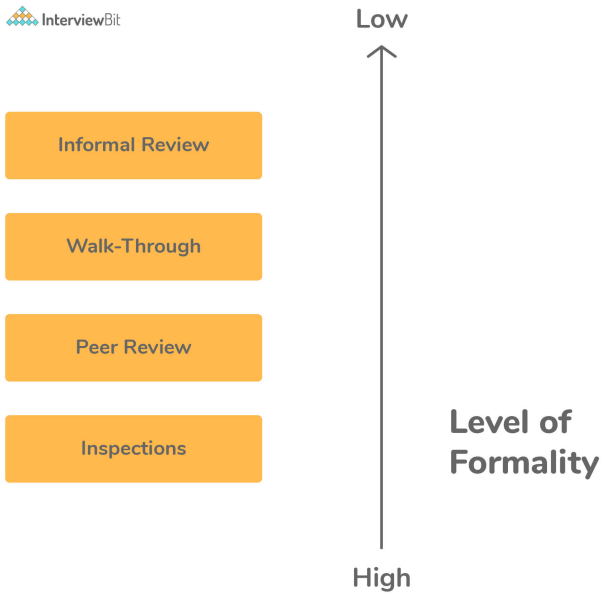
A unit test invokes the code and verifies the result with the expected result. If the expected and actual outcomes match, then the unit test passes. Otherwise, it fails.

A good unit test has the following characteristics:

1. It should test a single piece of functionality.
2. It is fully automated and repeatable.
3. It should run quickly and provide immediate feedback.
4. It should be isolated and shouldn't interact with external dependencies such as network, database, or file system unless needed. You can use the mocking technique to simulate the external dependencies and isolate the code under test.

17. What is static software testing?

Static testing is a technique in which you test the software without actually executing it. It involves doing code walkthroughs, code reviews, peer-reviews, or using sophisticated tools such as eslint, StyleCop to perform static analysis of the source code. Static testing is typically performed during software development.



18. What is dynamic software testing?

In contrast to static testing, dynamic software testing tests the software when it's executing. The tester runs the software in a test environment and goes through all the steps involved, entering the inputs and verifying the actual output with the expected result.

19. What is Quality Assurance(QA)?

QA stands for Quality Assurance. In a software development team, a QA ensures that the software is thoroughly tested before releasing it to the end-users.

Quality control is a software engineering process that is used to ensure the software quality and whether it implements the required standards and follows the protocols and procedures necessary.

In many software organizations, a tester and a QA can be the same person, but they can be different depending on the organization's size. The goal of the QA is to ensure quality in the shipped software.

20. What is a software bug?

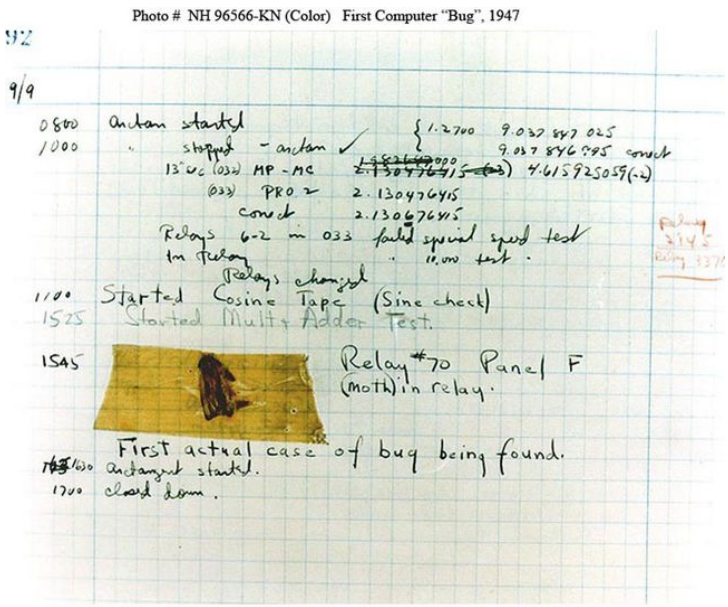
A software bug is an error in the software that produces wrong results. A software tester tests the software to find bugs in it.

There are many causes for the bugs—for example, poor design, sloppy programming, lack of version control, or miscommunication. Throughout development, developers introduce hundreds or thousands of bugs in the system. The goal of the tester is to uncover those bugs.

You can find a bug in many different ways, regardless of your role. When building the software, the software developer might notice the bug in another module, written by another developer or by themselves. The tester actively tries to find the bugs as part of a routine testing process. Finally, the users could see the bugs when the software is in production.

All bugs, no matter how they are found, are recorded into a bug-tracking system. A triage team triages the bugs and assigns a priority to the bug, and assigns the bug to a software developer to fix it. Once the developer resolves the problem, they check in the code and mark that bug as ready for testing. Once a bug is ready for testing, it goes to the tester, who tests the software to verify if it's indeed fixed. If it is, then it's closed. If not, they assign it to the same developer with a description of the exact steps to reproduce the bug. Some examples of popular bug-tracking systems include BugZilla, FogBugz, etc.

Trivia:
The first software bug was discovered by Admiral Grace Hopper, on September 9, 1947. After they opened a malfunctioning piece of hardware, they found an insect stuck in the relay. *Image Source: Link*



First Software Bug

Software Testing Interview Questions For Experienced

21. How much testing is sufficient? Or, is it possible to do exhaustive testing of the software?

It is impossible to exhaustively test software or prove the absence of errors, no matter how specific your test strategy is.

An extensive test that finds hundreds of errors doesn't imply that it has discovered them all. There could be many more errors that the test might have missed. The absence of errors doesn't mean there are no errors, and the software is perfect. It could easily mean ineffective or incomplete tests. To prove that a program works, you'd have to test all possible inputs and their combinations.

Consider a simple program that takes a string as an input that is ten characters long. To test it with each possible input, you'd have to enter 2610 names, which is impossible. Since exhaustive testing is not practical, your best strategy as a tester is to pick the test cases that are most likely to find errors. Testing is sufficient when you have enough confidence to release the software and assume it will work as expected.

22. Why developers shouldn't test the software they wrote?

Developers make poor testers. Here are some reasons why:

- They try to test the code to make sure that it works, rather than testing all the ways in which it doesn't work.
- Since they wrote it themselves, developers tend to be very optimistic about the software and don't have the correct attitude needed for testing: to break software.
- Developers skip the more sophisticated tests that an experienced tester would perform to break the software. They follow the happy path to execute the code from start to finish with proper inputs, often not enough to get the confidence to ship software in production.

However, it doesn't mean that developers shouldn't test the software before sending it to the tester. Developer testing helps find many bugs that are caused by programming errors. These are hard to find for a tester because they don't always have access to the source code.

23. What qualities a software tester should have?

Any software tester's goal is to find out as many bugs and problems in the system so that the customers don't have to. Hence, a good software tester should have a keen eye for detail. They should know the ins and outs of the software they are testing and push every aspect of the software to its limits, to identify bugs that are hard to find with the software's regular use.

Having the domain knowledge of the application is essential. If a tester doesn't understand the specific problems the software is trying to solve, they won't be able to test it thoroughly.

A good tester should keep the end-user in mind when they are testing. Having empathy with the end-user helps the tester ensure that the software is accessible and usable. Simultaneously, the tester should possess basic programming skills to think from a developer's perspective, which allows them to notice common programming errors such as null-references, out-of-memory errors, etc.

Communication, both written and verbal, is an essential skill for a tester. A tester will frequently have to interact with both the developers and the management. They should be able to explain the bugs and problems found during testing to the developers. For each bug found, a good tester should provide a detailed bug report consisting of all the information a developer would need to fix that problem. They should be able to make a good case to the management if they are uncomfortable releasing the software if it contains unresolved issues.

24. What is a bug report?

During testing, a tester records their observations, findings, and other information useful to the developers or the management. All this data belongs to a test record, also called a bug report.

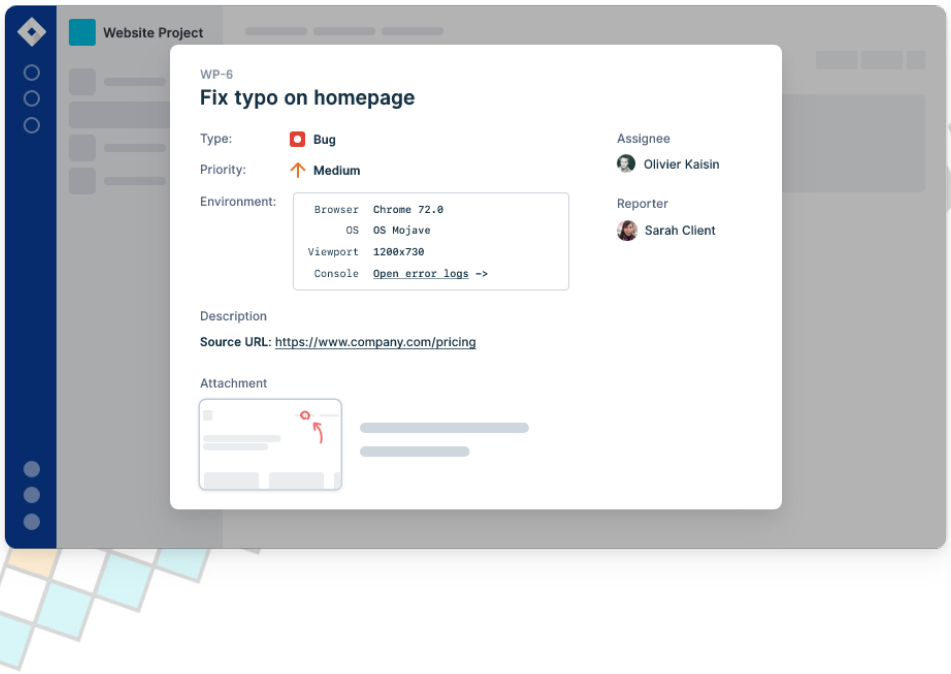
A detailed bug report is an important artifact produced during testing. It helps the team members with:

- Understand the problem,
- Steps to reproduce the problem,
- The environment and the specific conditions under which it happens, and
- The resolution if/when the developers fix the problem.

Here are a few bits of information that a good bug report should contain. *Image Source: Bugzilla*

Field	Description
Title	A short headline that summarizes the problem. It shouldn't be too long but just to give just the right information to the reader. It should be specific and accurate.
Description	The description should answer all the questions that are not explained by the title. It contains a detailed summary of the bug, its severity, and impact, steps to reproduce, expected results vs. the actual output.
Version	A lot of time can be wasted in trying to reproduce a bug in the wrong version of the product. Knowing the exact product version or the build number on which this bug was found is very useful to the developer in reproducing the bug.
Status	At any point, a bug can be either 'Active', 'Ready for Testing', or 'Closed'. A bug becomes active when it is found, is ready for testing once the developer fixes it. A tester can mark it closed if the developer fixed it, or active if not.
Steps to Reproduce	Though the steps to reproduce the problem can be provided in the description, sometimes having a distinct field force the tester to think about them. They include each step one must take to successfully reproduce the problem.
Assigned To	Name of the developer or the tester to whom this bug is assigned.

For example, here is a picture of a bug reported on Jira, a popular bug-tracking software.



25. What are some important testing metrics?

Testing metrics provide a high-level overview to the management or the developers on how the project is going and the next action steps.

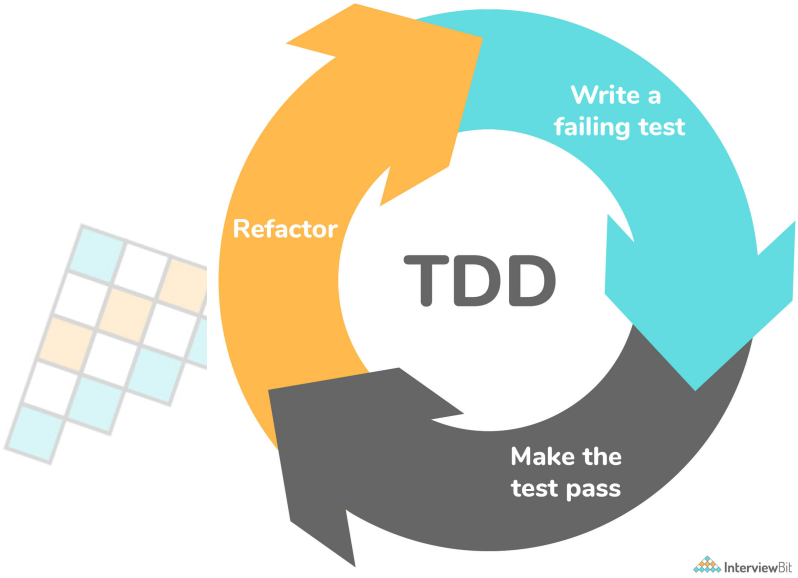
Here are some of the metrics derived from a record of the tests and failures:

- Total number of defects found, ordered by their severity
- Total number of bugs fixed
- Total number of problems caused by an error in the source code vs. configuration or external environmental factors
- Bug find and fix rate over time
- Bugs by produce/feature area
- The average time is taken by a bug since it's found and fixed.
- Total time spent on new feature development vs. time spent on resolving bugs and failures
- Number of outstanding bugs before a release
- Bugs/failures reported by the customers vs. those found by the testers

26. What is Test-Driven-Development?

Test-Driven-Development (TDD) is a popular software development technique, first introduced by Kent Beck in his book with the same name, published in 1999.

In TDD, a developer working on a feature first writes a failing test, then writes just enough code to make that test pass. Once they have a passing test, they add another failing test and then write just enough code to pass the failing test. This cycle repeats until the developer has the fully working feature. If the code under the test has external dependencies such as database, files, or network, you can mock them to isolate the code.



Benefits of TDD:

- Writing tests first forces you to think about the feature you are trying to build, helping you produce better code.
- As you always have a working set of tests at hand, a failing test indicates that the problem is with the code you just added, reducing the time spent in debugging.
- Writing tests help the developer to clarify the requirements and specification. It's challenging to write good tests for a poor set of requirements.
- It's tough to produce high-quality software unless you can test the software after each new change. You can never be sure that your new code didn't break the working software. TDD gives you the confidence to add new code, as you already have a test in place.

27. What is Selenium? What are its benefits?

Selenium is a web browser automation tool that automates the test suits you need to run on a web browser.

Some of the benefits of Selenium include:

- It is open-source software, eliminating licensing costs.
- It supports all the major languages, such as Java, C#, Python, Ruby, etc.
- It supports all the major web browsers, e.g., Google Chrome, Firefox, Safari, etc.
- You can integrate it with other testing frameworks and tools to build a comprehensive test suite for your software.

28. What is boundary value analysis?

In software, many errors occur near the edges of the range of the data values. For example, when the programmer uses the greater-than operator (>) instead of the greater-than-or-equal-to (>=) operator, it causes the off-by-one indexing error.

Typically, developers miss these boundary cases because they follow a happy path when developing and testing. Boundary value analysis helps to discover the errors caused by extreme values. The tester chooses the test data at and immediately above and below the boundaries of the input domain of the data.

For example, if an input field expects a string of 20 characters long, the tester tests it with strings of lengths 19, 20, and 21.

29. What are the differences between manual and automated testing?

Manual Testing	Automated Testing
A human tester tests the software by manually running the test cases and observing and comparing the actual and expected outputs.	A tester or a programmer uses scripts and tools that execute the software and compares the actual and expected outputs.
Manual testing is not reproducible and repeatable.	Since it is programmed, automated testing is consistently reproducible and repeatable. It can be executed as many times as the tester wants.
For new features, a tester can quickly test the feature manually, without much configuration and setup.	To set up automated testing, there's the initial investment required to write the tests and prepare an environment to run those tests on.
Manual testing is useful for finding bugs in the user interface or accessibility issues.	Automated testing is more suitable for catching bugs that a human tester would miss, such as programming bugs, business logic errors.
Manual testing is prone to human errors and is slow.	As there is no human participation involved (other than writing tests), automated testing is more reliable. It is much faster than manual testing.

30. What is a user story?

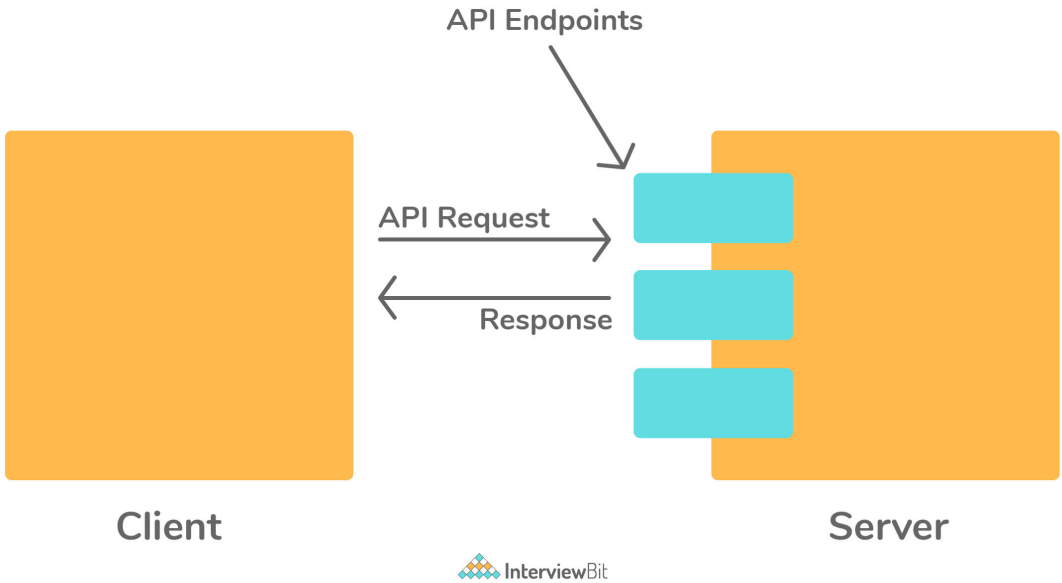
All software has a target user. A user story describes the user's motivations and what they are trying to accomplish by using the software. Finally, it shows how the user uses the application. It ignores the design and implementation details.

A user story aims to focus on the value provided to the end-user instead of the exact inputs they might enter and the expected output.

In a user story, the tester creates user personas with real names and characteristics and tries to simulate a real-life interaction with the software. A user story often helps fish out hidden problems that are often not revealed by more formal testing processes.

31. What is an API?

API stands for Application Programming Interface. It is a means of communication between two software components. An API abstracts the internal workings and complexity of a software program and allows the user of that API to solely focus on the inputs and outputs required to use it.

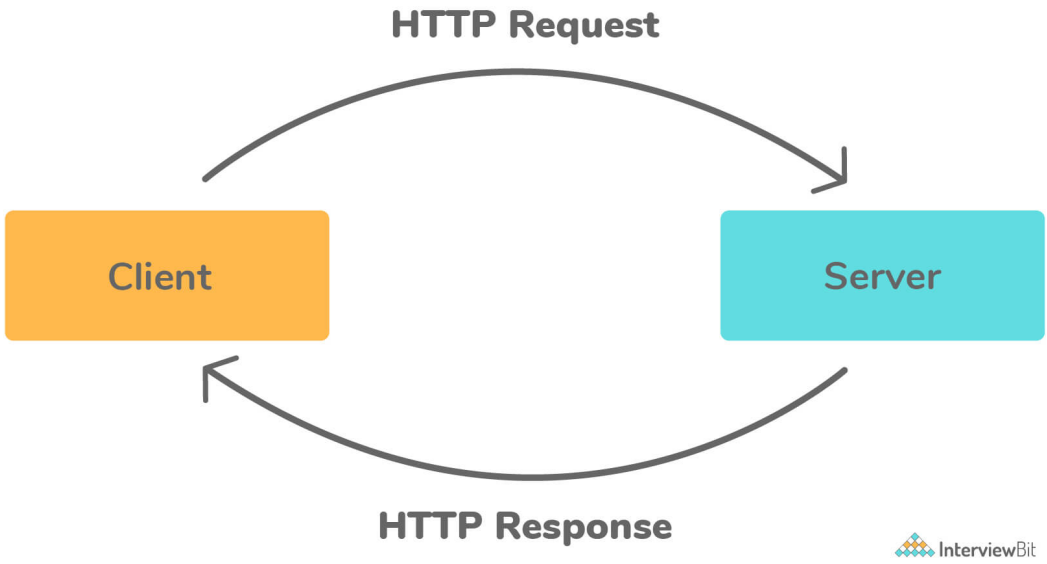


When building software, developers rarely write software from scratch and make use of other third-party libraries. An API allows two software components to talk to each other by providing an interface that they can understand.

Another use of an API is to provide data required by an application. Let's say you are building a weather application that displays the temperature. Instead of building the technology to collect the temperature yourself, you'd access the API provided by the meteorological institute.

32. What are the different HTTP status codes that a server can return?

An HTTP status code is a three-digit number that indicates the status of an incoming HTTP request, that is, if the request has been completed or not.



A server can send the following five types of responses for an HTTP request.

1. Information (100 - 199): These status codes provide a temporary response. The response consists of the status line and optional headers and terminates by an empty line.
2. Success (200 - 299): Indicate that the incoming HTTP request was successfully received, understood, and accepted.
3. Redirect (300 - 399): These status codes indicate further actions the client should take to satisfy the HTTP request. It can mean that the requested resource may have moved temporarily or permanently. It can also redirect the client to another URL.
4. A client error (400 - 499): Indicate a problem with the client who initiated the HTTP request.
5. Server error (500 - 599): The 5XX status code indicates a problem on the server while processing the request.

33. What is API testing?

API testing ensures that the APIs that the software is using work as expected. The tester writes code that makes an API request to the server that provides the API, provides the required inputs, collects the output from the response, and matches the actual output with the expected output.

API testing primarily concerns the business logic of the software that's exposing the API. It does not involve the look and feel, accessibility, or usability of the software. API testing can be automated to make it repeatable and reproducible each time they run.

34. What is a test environment?

A test environment consists of a server/computer on which a tester runs their tests. It is different from a development machine and tries to represent the actual hardware on which the software will run; once it's in production.

Whenever a new build of the software is released, the tester updates the test environment with the latest build and runs the regression tests suite. Once it passes, the tester moves on to testing new functionality.

35. What is test coverage?

Test coverage is a metric that indicates how much of the source code is covered by the tests, allowing the tester to verify the quality of their testing. It helps the tester figure out whether they are testing everything they're supposed to test.

Test coverage can mean different things to different people, depending on the particulars of their testing approaches.

1. Product: It means looking at test coverage to answer the question: Which features or the areas of the software does your tests cover?
2. Requirements: The software might work well, but it's not useful to the customer if it doesn't satisfy their needs. Requirements coverage indicates how many of the requirements are tested.
3. Source Code: This is usually a developer's domain and is a white-box testing technique. The developer can check how much of their source code is covered by the unit tests.

36. What is meant by browser automation?

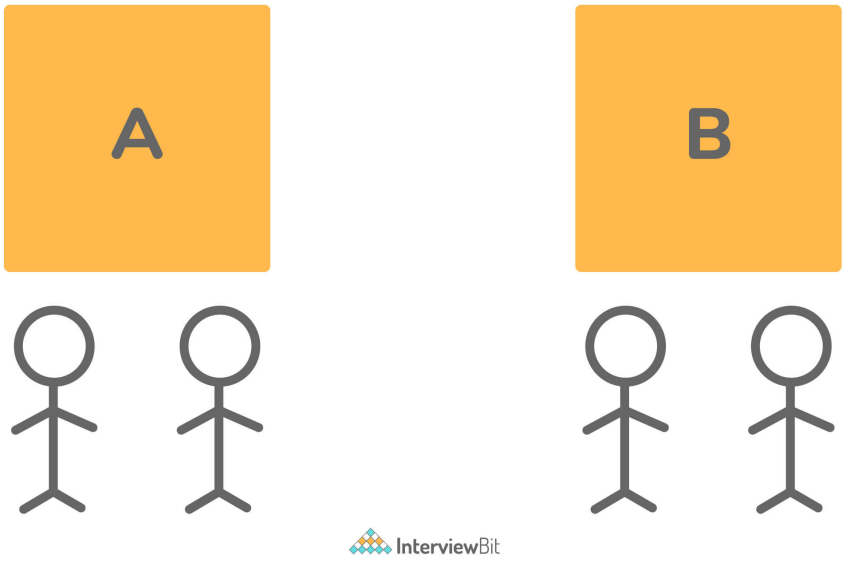
It’s a process of automatically testing a web application’s functionality in a browser, where a program launches the browser, navigates to the application, and interacts with the user interface by clicking buttons or links, just like an average user would.

The only difference is that the browser automation can test this very quickly and often, whereas the same test would take a human tester a long time. It’s part of automated testing. Some essential tools for browser testing include Selenium, protractor.js, and cypress.

37. What is A/B testing?

A/B testing is the process of testing two or more different versions of your software with users to assess which performs better. It is a low-risk way of testing variations of a new or existing functionality.

You can choose a part of your users to use feature A. The other group uses feature B. Then user feedback and response are evaluated using statistical testing to decide the final version of the feature.



Typically, A/B testing is used to test the user experience of different interfaces. This allows the team to quickly gather feedback and test their initial hypothesis.

38. What is meant by Code Coverage?

Code coverage is one of the important testing metrics. It indicates the ratio of the codebase under unit tests to the entire codebase. Code coverage of 50% means that the unit tests cover half of the codebase.

It is crucial to keep in mind that 100% code coverage doesn't mean that the software is error-free or there are no bugs. It just means that the unit tests cover all the code. However, it's still possible that the tests don't test all branches that code could take or the problems with business logic execution.

39. List some of the popular testing tools/frameworks, providing a brief description of each.

1. Selenium: a web browser automation tool that automates the test suites you need to run on a web browser.[1]
2. Protractor: An end-to-end test framework for Angular and AngularJS applications. Protractor runs tests against your application running in a real browser, interacting with it as a user would.[2]
3. Cypress: A modern front-end testing tool built for the modern web. Though it's similar to Selenium and Protractor, it's architecturally different from them.[3]
4. Jasmine: This is an open-source JavaScript testing framework that allows you to write behavior-driven tests.[4]
5. JUnit and NUnit: These are unit testing frameworks for Java and C# programming languages, respectively.

40. What are the different types of severity you can assign to a bug?

Though it varies depending on the size and structure of the software development teams, typically, a bug can be assigned the following types of severities, going from low to high:

- Low
  - User Interface bugs
  - Accessibility issues
- Medium
  - Leaky abstractions
  - Software hangs
  - Users unable to perform a specific action
  - Boundary conditions
- High
  - Crashing under high load
  - Business logic and/or calculation errors
  - Any user action that causes the software to crash
  - Exposing sensitive user data
  - Security problems
  - Loss of data

Conclusion

41. Conclusion

Software testing is an important activity that ensures quality, giving the confidence to release the software to customers. This article explained the testing process and its importance in software development. It also explained different types of testing, such as functional and non-functional testing, white-box and black-box testing, and the benefits of each.

However, testing is only a single component of a good software development strategy. A development team should use high coding standards, best practices, and patterns to reduce the bug count. As a long-term strategy, the best way to improve the testing process is to test frequently, measure the results, gather feedback and use it to get better.

References:

- Selenium: https://www.selenium.dev/documentation
- Protractor: https://www.protractortest.org/
- Cypress: https://docs.cypress.io/
- Jasmine: https://jasmine.github.io/

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