**Different Environment in an organization:**

* Development
* QA == Functional testing of the system
* System Integration Testing == Tests the system from end to end
* SAT = > Site Acceptance Testing => Site Acceptance Testing is the stage where the customer conducts testing for the components supplied under the project scope and tests the conformance of the delivered solution to the Solution Definition Document and functional specifications.
* UAT= > User Acceptance Testing = Allows the user to validate the functionality over time
* Production == Production
* Production Parallel == A parallel of production to replicate production issues
* CCE = Client Certification Environment

**Types of Testing**

* Continuous integration, delivery, and deployment all rely heavily on automated tests to determine the efficacy and correctness of each code change.
* Different types of tests are needed throughout these processes to gain confidence in a given solution.
* While the categories below in no way represent an exhaustive list, and although there is disagreement on the exact definition of each type, these broad categories of tests represent a variety of ways to evaluate code in different contexts.

**Smoke Testing**

* Smoke tests are a special kind of initial checks designed to ensure very basic functionality as well as some basic implementation and environmental assumptions. Smoke tests are generally run at the very start of each testing cycle as a sanity check before running a more complete test suite.
* The idea behind this type of test is to help to catch big red flags in an implementation and to bring attention to problems that might indicate that further testing is either not possible or not worthwhile. Smoke tests are not very extensive, but should be extremely quick. If a change fails a smoke test, its an early signal that core assertions were broken and that you should not devote any more time to testing until the problem is resolved.
* Context-specific smoke tests can be employed at the start of any new phase testing to assert that the basic assumptions and requirements are met. For instance, smoke tests can be used both prior to integration testing or deploying to staging servers, but the conditions to be tested will vary in each case.

**Unit Testing**

* Unit tests are responsible for testing individual elements of code in an isolated and highly targeted way. The functionality of individual functions and classes are tested on their own. Any external dependencies are replaced with stub or mock implementations to focus the test completely on the code in question.
* Unit tests are essential to test the correctness of individual code components for internal consistency and correctness before they are placed in more complex contexts. The limited extent of the tests and the removal of dependencies makes it easier to hunt down the cause of any defects. It also is the best time to test a variety of inputs and code branches that might be difficult to hit later on. Often, after any smoke tests, unit tests are the first tests that are run when any changes are made.
* Unit tests are typically run by individual developers on their own work station prior to submitting changes. However, continuous integration servers almost always run these tests again as a safe guard before beginning integration tests.

**Integration Testing**

* After unit tests, integration testing is performed by grouping together components and testing them as an assembly. While unit tests validate the functionality of code in isolation, integration tests ensure that components cooperate when interfacing with one another. This type of testing has the opportunity to catch an entirely different class of bugs that are exposed through interaction between components.
* Typically, integration tests are performed automatically when code is checked into a shared repository. A continuous integration server checks out the code, performs any necessary build steps (usually performing a quick smoke test to make sure the build was successful) and then runs unit and integration tests. Modules are hooked together in different combinations and tested.
* Integration tests are important for shared work because they protect the health of the project. Changes must prove that they do not break existing functionality and that they interact with other code as expected. A secondary aim of integration testing is to verify that the changes can be deployed into a clean environment. This is frequently the first testing cycle that is performed off of the developer's own machines, so unknown software and environmental dependencies can also be discovered during this process. This is usually also the first time that new code is tested against real external libraries, services, and data.

**System Testing**

* Once integration tests are performed, another level of testing called system testing can begin. In many ways, system testing acts as an extension to integration testing. The focus of system tests are to make sure that groups of components function correctly as a cohesive whole.
* Instead of focusing on the interfaces between components, system tests typically evaluate the outward functionality of a full piece of software. This set of tests ignores the constituent parts in order to gauge the composed software as a unified entity. Because of this distinction, system tests usually focus on user- or externally-accessible interfaces.

**Acceptance Testing**

* Acceptance tests are one of the last tests types that are performed on software prior to delivery. Acceptance testing is used to determine whether a piece of software satisfies all of the requirements from the business or user's perspective. These tests are sometimes built against the original specification and often test interfaces for the expected functionality and for usability.
* Acceptance testing is often a more involved phase that might extend past the release of the software. Automated acceptance testing can be used to make sure the technological requirements of the design were met, but manual verification also usually plays a role.
* Frequently, acceptance testing begins by deploying the build to a staging environment that mirrors the production system. From here, the automated test suites can be run and internal users can access the system to check whether it functions the way they need it to. After release or offering beta access to customers, further acceptance testing is performed by evaluating how the software functions with real use and by collecting feedback from users.

**System Acceptance Testing (SAT):**

It is end-to-end testing wherein testing environment is similar to the production environment. We can also called it End to End testing. Here, we navigate through all the features of the software and test if the end business / and feature works. We just test the end feature and don’t check for data flow or do functional testing and all.

**User Acceptance Testing (UAT):**

Acceptance testing is done by end users. Here, they use the s/w for the business for a particular period of time and check whether the s/w can handle all kinds of real-time business scenarios / situations.

**What is Continuous Testing?**

I will advise you to follow the below mentioned explanation:

Continuous Testing is the process of executing automated tests as part of the software delivery pipeline to obtain immediate feedback on the business risks associated with in the latest build.

In this way, each build is tested continuously, allowing Development teams to get fast feedback so that they can prevent those problems from progressing to the next stage of Software delivery life-cycle.

This dramatically speeds up a developer’s workflow as there’s no need to manually rebuild the project and re-run all tests after making changes.

**What is Automation Testing?**

Automation testing or Test Automation is a process of automating the manual process to test the application/system under test. Automation testing involves use of separate testing tools which lets you create test scripts which can be executed repeatedly and doesn’t require any manual intervention.

**What are the benefits of Automation Testing?**

I have listed down some advantages of automation testing. Include these in your answer and you can add your own experience of how Continuous Testing helped your previous company:

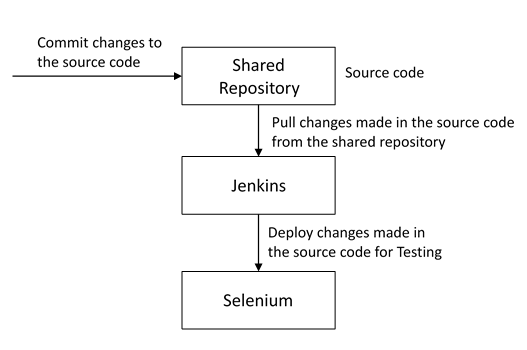
* Supports execution of repeated test cases
* Aids in testing a large test matrix
* Enables parallel execution
* Encourages unattended execution
* Improves accuracy thereby reducing human generated errors
* Saves time and money

**How to automate Testing in DevOps lifecycle?**

I have mentioned a generic flow below which you can refer to:

In DevOps, developers are required to commit all the changes made in the source code to a shared repository. Continuous Integration tools like Jenkins will pull the code from this shared repository every time a change is made in the code and deploy it for Continuous Testing that is done by tools like Selenium as shown in the below diagram.

In this way, any change in the code is continuously tested unlike the traditional approach.



**Why is Continuous Testing important for DevOps?**

You can answer this question by saying, “Continuous Testing allows any change made in the code to be tested immediately. This avoids the problems created by having “big-bang” testing left to the end of the cycle such as release delays and quality issues. In this way, Continuous Testing facilitates more frequent and excellent quality releases.”

**What are the key elements of Continuous Testing tools?**

Key elements of Continuous Testing are:

* **Risk Assessment:** It Covers risk mitigation tasks, technical debt, quality assessment and test coverage optimization to ensure the build is ready to progress toward next stage.
* **Policy Analysis:** It ensures all processes align with the organization’s evolving business and compliance demands are met.
* **Requirements Traceability:** It ensures true requirements are met and rework is not required. An object assessment is used to identify which requirements are at risk, working as expected or require further validation.
* **Advanced Analysis:** It uses automation in areas such as static code analysis, change impact analysis and scope assessment/prioritization to prevent defects in the first place and accomplishing more within each iteration.
* **Test Optimization:** It ensures tests yield accurate outcomes and provide actionable findings. Aspects include Test Data Management, Test Optimization Management and Test Maintenance
* **Service Virtualization:** It ensures access to real-world testing environments. Service visualization enables access to the virtual form of the required testing stages, cutting the waste time to test environment setup and availability.

**Which Testing tool are you comfortable with and what are the benefits of that tool?**

I have mentioned an example below:

I have worked on Selenium to ensure high quality and more frequent releases.

Some advantages of Selenium are:

* It is free and open source
* It has a large user base and helping communities
* It has cross Browser compatibility (Firefox, chrome, Internet Explorer, Safari etc.)
* It has great platform compatibility (Windows, Mac OS, Linux etc.)
* It supports multiple programming languages (Java, C#, Ruby, Python, Pearl etc.)
* It has fresh and regular repository developments
* It supports distributed testing

**What are the Testing types supported by Selenium?**

Selenium supports two types of testing:

**Regression Testing**: It is the act of retesting a product around an area where a bug was fixed.  
**Functional Testing**: It refers to the testing of software features (functional points) individually.

**What is Selenium IDE?**

It is an integrated development environment for Selenium scripts. It is implemented as a Firefox extension, and allows you to record, edit, and debug tests.

Selenium IDE includes the entire Selenium Core, allowing you to easily and quickly record and play back tests in the actual environment that they will run in.

With autocomplete support and the ability to move commands around quickly, Selenium IDE is the ideal environment for creating Selenium tests no matter what style of tests you prefer.

**What is the difference between Assert and Verify commands in Selenium?**

* Assert command checks whether the given condition is true or false. Let’s say we assert whether the given element is present on the web page or not. If the condition is true, then the program control will execute the next test step. But, if the condition is false, the execution would stop and no further test would be executed.
* Verify command also checks whether the given condition is true or false. Irrespective of the condition being true or false, the program execution doesn’t halt i.e. any failure during verification would not stop the execution and all the test steps would be executed.

**How to launch Browser using WebDriver?**

The following syntax can be used to launch Browser:

**WebDriver driver = new FirefoxDriver();**  
**WebDriver driver = new ChromeDriver();**  
**WebDriver driver = new InternetExplorerDriver();**

**When should I use Selenium Grid?**

It can be used to execute same or different test scripts on multiple platforms and browsers concurrently to achieve distributed test execution. This allows testing under different environments and saving execution time remarkably.