## **Software Development Life Cycle**

The SDLC process includes planning, designing, developing, testing and deploying with ongoing maintenance to create and manage applications efficiently.

When faced with the task of producing high-quality software that meets a client’s expectations, requirements, time-frame, and cost estimations; understanding the SDLC is crucial.

**“SDLC methodologies”** are used to create complex applications of varying sizes and scales, such as Agile, Waterfall and Spiral. Each model follows a particular life cycle in order to ensure success in the process of software development.

### **SDLC Phases:**

* Planning
* Requirements
* Designing
* Developing
* Testing
* Deploying
* Maintenance

#### **Stage 1: Project Planning**

The first stage of SDLC is all about “What do we want?” Project planning is a vital role in the software delivery lifecycle since this is the part where the team estimates the cost and defines the requirements of the new software.

#### **Stage 2: Gathering Requirements & Analysis**

The second step of SDLC is gathering maximum information from the client requirements for the product by Business analyst. Discuss each detail and specification of the product with the customer. The development team will then analyse the requirements keeping the design and code of the software in mind.

The main goal of this stage is that everyone understands even the minute detail of the requirement. Hardware, operating systems, programming, and [security](https://www.betsol.com/security-services/) are to name the few requirements.

#### **Stage 3: Designing**

In the design phase (3rd step of SDLC), the program developer scrutinizes whether the prepared software suffices all the requirements of the end-user. Additionally, if the project is feasible for the customer technologically, practically, and financially. Once the developer decides on the best design approach, he then selects the program languages like Oracle, [Java](https://www.betsol.com/blog/java-memory-management-for-java-virtual-machine-jvm/), etc., that will suit the software.

Once the design specification is prepared, all the stakeholders will review this plan and provide their feedback and suggestions. It is absolutely mandatory to collect and incorporate stakeholder’s input in the document, as a small mistake can lead to cost overrun.

#### **Stage 4: Coding or Implementation**

Time to code! It means translating the design to a computer-legible language. In this fourth stage of SDLC, the tasks are divided into modules or units and assigned to various developers. The developers will then start building the entire system by writing code using the programming languages they chose.

This stage is considered to be one of the longest in SDLC. The developers need certain predefined coding guidelines, and programming tools like interpreters, compilers, debugger to implement the code.

The developers can show the work done to the business analysts in case if any modifications or enhancements required.

#### **Stage 5: Testing**

Once the developers build the software, then it is deployed in the testing environment. Then the testing team tests the functionality of the entire system. In this fifth phase of SDLC, the testing is done to ensure that the entire application works according to the customer requirements.

After testing, the [QA and testing](https://www.betsol.com/software-development-and-testing/) team might find some bugs or defects and communicate the same with the developers. The development team then fixes the bugs and send it to QA for a re-test. This process goes on until the software is stable, bug-free and working according to the business requirements of that system.

#### **Stage 6: Deploying**

The sixth phase of SDLC: Once the testing is done, and the product is [ready for deployment](https://www.betsol.com/blog/how-to-make-software-deployments-easier/), it is released for customers to use. The size of the project determines the complexity of the deployment. The users are then provided with the training or documentation that will help them to operate the software.

Again, a small round of testing is performed on production to ensure environmental issues or any impact of the new release.

#### **Stage 7: Maintenance**

The actual problem starts when the customer actually starts using the developed system and those needs to be solved from time to time. Maintenance is the seventh phase of SDLC where the developed product is taken care of.

According to the changing user end environment or technology, the software is updated timely.

### **Predominant Models of SDLC:**

#### **Waterfall Model:** This SDLC model is considered to be the oldest and most forthright. We finish with one phase and then start with the next, with the help of this methodology. Each of the phases in this model has its own mini-plan and each stage waterfalls into the next. A drawback that holds back this model is that even the small details left incomplete can hold an entire process.

#### **Agile Model**: Agile is the new normal; It is one of the most utilized models, as it approaches software development in incremental but rapid cycles, commonly referred to as “sprints”. With new changes in scope and direction being implemented in each sprint, the project can be completed quickly with higher flexibility. Agile means spending less time in the planning phases, and a project can diverge from original specifications.

#### **Iterative Model**: This SDLC model stresses on repetition. Developers create a version rapidly for relatively less cost, then test and improve it through successive versions. One big disadvantage of this model is that if left unchecked, it can eat up resources fast.

#### **V-Shaped Model**: This model can be considered as an extension of the waterfall model, as it includes tests at each stage of development. Just like the case with waterfall, this process can run into obstructions.

#### **Big Bang Model**: This SDLC model is considered best for small projects as it throws most of its resources at development. It lacks the detailed requirements definition stage when compared to the other methods.

#### **Spiral Model**: One of the most flexible of the SDLC models is the spiral model. It resembles the iterative model in its emphasis on repetition. Even this model goes through the planning, design, build and test phases again and again, with gradual improvements at each stage.

### **Wrapping-up SDLC:**

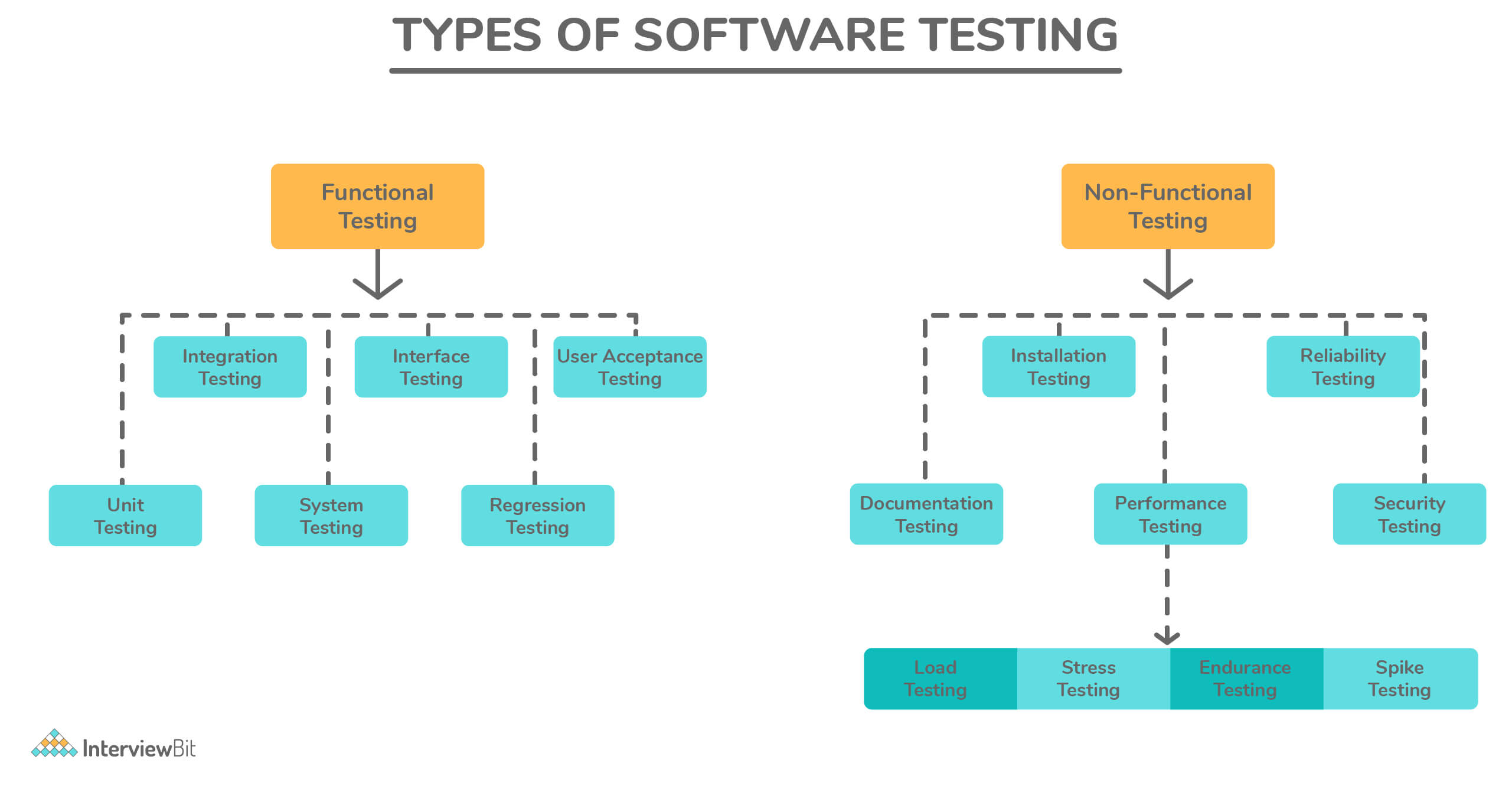
SDLC can be a great tool that can help us with the highest level of documentation and management control. But failure to consider customer’s requirements, users or stakeholders can lead to project failure.

## **Software Testing**

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.

### **Different types of testing:**



| **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. 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. |
| 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. |
| Stress Testing | Ensures that the performance of the software doesn’t degrade when the load increases. In stress testing, the tester subjects the software under heavy loads, such as a high number of requests or stringent memory conditions to verify if it works well. |
| Usability Testing | Measures how usable the software is. This is typically performed with a sample set of end-users, who use the software and provide feedback on how easy or complicated it is to use the software. |
| Security Testing | Now more important than ever. Security testing tries to break a software’s security checks, to gain access to confidential data. Security testing is crucial for web-based applications or any applications that involve money. |

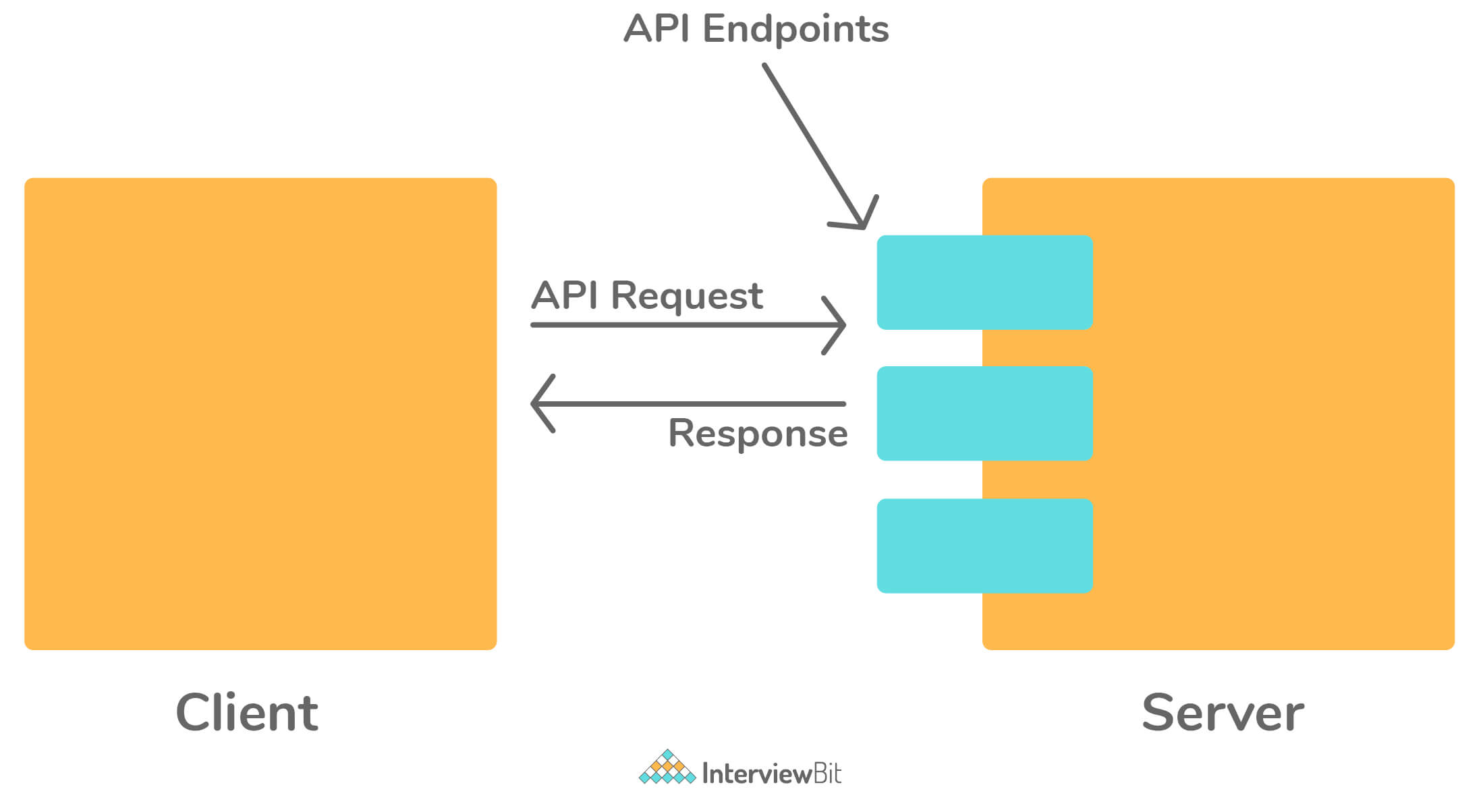
### **Principles of software testing: (TEDAPET)**

Software testing is governed by seven principles:

* **Beware of Absence-of-errors fallacy:**Even if the software is 99% bug-free, it is unusable if it does not conform to the user's requirements. Software needs to be bug-free 99% of the time, and it must also meet all customer requirements and user friendly.
* **Testing shows the presence of defects, not their absence:** Testing can verify the presence of defects in software, but it cannot guarantee that the software is defect-free. Testing can minimize the number of defects, but it can't remove them all.
* **Exhaustive testing is not possible:**The software cannot be tested exhaustively, which means all possible test cases cannot be covered. Testing can only be done with a select few test cases, and it's assumed that the software will produce the right output in all cases. Taking the software through every test case will cost more, take more effort, etc., which makes it impractical.
* **Defects cluster together:** The majority of defects are typically found in a small number of modules in a project. According to the **Pareto Principle**, 80% of software defects arise from 20% of modules.
* **Beware of the Pesticide Paradox:**It is impossible to find new bugs by re-running the same test cases over and over again. Thus, updating or adding new test cases is necessary in order to find new bugs. Preferably, use a code-coverage tool to ensure that your test cases cover all code paths.
* **Early testing saves time and money:** Early testing is crucial to finding the defect in the software. In the early stages of SDLC, defects will be detected more easily and at a lower cost. Software testing should start at the initial phase of software development, which is the requirement analysis phase.
* **Testing is context-dependent:**The testing approach varies depending on the software development context. Software needs to be tested differently depending on its type. For instance, an ed-tech site is tested differently than an Android app.

### **API (Application Programming Interface):**

API 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.

### **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.

### **Code coverage testing:**

When software is being tested, the code coverage measures how much of the program's source code is covered by the test plan. Code coverage testing runs in parallel with actual product testing.

Using the code coverage tool, you can monitor the execution of statements in your source code. A complete report of the pending statements, along with the coverage percentage, is provided at the end of the final testing.

Among the different types of test coverage techniques are:

* **Statement/Block Coverage:** Measures how many statements in the source code have been successfully executed and tested.
* **Decision/Branch Coverage:** This metric measures how many decision control structures were successfully executed and tested.
* **Path Coverage:**This ensures that the tests are conducted on every possible route through a section of the code.
* **Function coverage:**It measures how many functions in the source code have been executed and tested at least once.

### **Box Testing:**

* **Black-box testing in software testing**: In black-box testing, the system is tested only in terms of its external behaviour; it does not consider how the software functions on the inside. This is the only limitation of the black-box test. It is used in Acceptance Testing and System Testing.
* **White-box testing in software testing:** A white-box test is a method of testing a program that takes into account its internal workings as part of its review. It is used in integration testing and unit testing.
* **Grey-box testing in software testing:** A Gray Box Testing technique can be characterized as a combination of a black box as well as a white box testing technique used in the software testing process. Using this technique, you can test a software product or application with a partial understanding of its internal structure.

### **Test scenarios, Test scripts, and Test cases in software testing:**

* **Test Case:**Test Cases are a series of actions executed during software development to verify a particular feature or function. A test case consists of test steps, test data, preconditions, and postconditions designed to verify a specific requirement.
* **Test Scenario:** Usually, a test scenario consists of a set of test cases covering the end-to-end functionality of a software application. A test scenario provides a high-level overview of what needs to be tested.
* **Test Scripts:**When it comes to software testing, a test script refers to the set of instructions that will be followed in order to verify that the system under test performs as expected. The document outlines each step to be taken and the expected results.

Test Case is a manual approach of software testing. Test Script is an automatic approach of software testing.

### **Test Plan:**

A test plan is basically a dynamic document monitored and controlled by the testing manager. The success of a testing project totally depends upon a well-written test plan document that describes software testing scope and activities. It basically serves as a blueprint that outlines the what, when, how, and more of the entire test process.

A test plan must include the following details:

* Test Strategy
* Test Objective
* Test Scope
* Reason for Testing
* Exit/Suspension Criteria
* Resource Planning
* Test Deliverables.

### **Test Report:**

Test report is basically a document that includes a total summary of testing objectives, activities, and results. It is very much required to reflect testing results and gives an opportunity to estimate testing results quickly. It helps us to decide whether the product is ready for release or not. It also helps us determine the current status of the project and the quality of the product. A test report must include the following details:

* Test Objective
* Project Information
* Defect
* Test Summary

### **Test Deliverables:**

Test deliverables, also known as test artifacts, are basically a list of all of the documents, tools, and other components that are given to the stakeholders of a software project during the SDLC. Test deliverables are maintained and developed in support of the test. At every phase of SDLC, there are different deliverables as given below:

**Before Testing Phase**

* Test plans document.
* Test cases documents
* Test Design specifications.

**During Testing Phase**

* Test Scripts
* Simulators.
* Test Data
* Test Traceability Matrix
* Error logs and execution logs

**After testing Phase**

* Test Results/reports
* Defect Report
* Installation/ Test procedures guidelines
* Release notes

### **Different categories of debugging:**

* Brute force debugging
* Backtracking
* Cause elimination
* Program slicing
* Fault tree analysis

### **Common mistakes that lead to major issues:**

* Poor Scheduling
* Underestimating
* Ignoring small issues
* Not following the exact process
* Improper resource allocation

### **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.

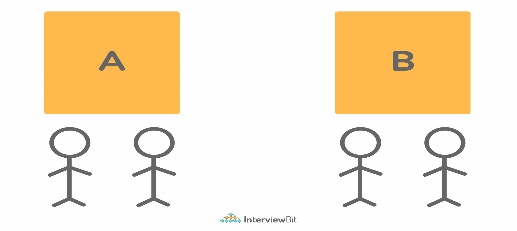
### **Popular software testing tools:**

1. Selenium: a web browser automation tool that automates the test suites you need to run on a web browser.
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.
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.
4. Jasmine: This is an open-source JavaScript testing framework that allows you to write behaviour-driven tests.
5. JUnit and NUnit: These are unit testing frameworks for Java and C# programming languages, respectively.

### **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.

### **Defects in software testing:**

The term defect refers to a system error that prevents the intended action from being accomplished. Testing is most important when it comes to finding defects. Testing needs to begin early in the development process since defects can be found throughout. As shown in the following figure, defects are divided into three main categories:

* **Wrong:** It implies incorrect implementation of requirements. There is a variance between the specifications and what was expected, resulting in this defect.
* **Missing:** This indicates that a specification has not been implemented, or a requirement of the customer has not been properly noted.
* **Extra:**In this case, the defect is caused by a requirement incorporated into the product that was not provided by the end-user.

### **SPICE in software testing:**

SPICE stands for Software Process Improvement and Capability Determination. In the field of software development processes, SPICE is a standard framework for assessing the efficiency and effectiveness of the development process. IEC (International Electrotechnical Commission) and ISO (International Organization for Standardization) jointly developed SPICE.

### **Latent defect and Masked defect:**

* **Latent Defect:**Latent defects are defects that exist but have not yet been invoked because the conditions required to invoke them have not been met. As a systematic flaw, it encompasses the entire production process of the software, including all pre-production testing and extended testing. When users perform a particular task in an unusual or rare situation or without the presence of usual scenarios, latent defects are revealed.
* **Masked Defect:** These are the defects that have not yet resulted in a failure since another defect hides that portion of the code from being executed. It can only be discovered when the defect hiding it is exposed by the user through a specific operation. There are defects that are hidden or marked by another defect and remain hidden until the other defect is detected.

### **Sanity testing:**

The term 'sanity testing' refers to a subset of regression testing. The sanity testing ensures that the changes made to the code do not adversely affect the system's performance. After the software build is received, a sanity test is conducted to ensure that the changes made to the code are working correctly.

As a checkpoint, this testing is used to determine whether the build can proceed with further testing. Sanity testing focuses on validating the functionality of the application rather than detailed testing.

### **TestNG:**

The TestNG framework for Java is an open-source advanced test automation framework that is designed to benefit both testers and developers. The purpose of TestNG is to provide an easy-to-use, readable, structured, maintainable, and user-friendly environment for automated tests. NG stands for 'Next Generation' in TestNG.

The high-end annotations, such as data providers, make cross-browser testing easier since you can test across multiple devices and browsers. Furthermore, the framework has an inbuilt mechanism for handling exceptions that prevent the program from terminating unexpectedly.

You can skip a particular test method or code by setting the 'enabled' parameter to ‘false’ in test annotations.

@Test(enabled = false).

Prioritizing the order of your test methods can be accomplished by defining a priority order. Consequently, the test will execute in accordance with the priority set.

@Test(priority=2)

### **Object Repository:**

Object Repository is a collection of web elements and their locators that belong to the Application Under Test (AUT). The QAs maintain all element locators in a separate file known as the property file (. properties) in Selenium. During execution, it serves as a means of identifying objects between the test script and the application.

### **Valuable steps to resolve issues while testing:**

* Record: Keep track of any problems that arise and resolve them.
* Report: Inform higher-level managers of the issues.
* Control: Establish a process for managing issues.

### **Qualities of a software tester:**

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.

### **Boundary value analysis:**

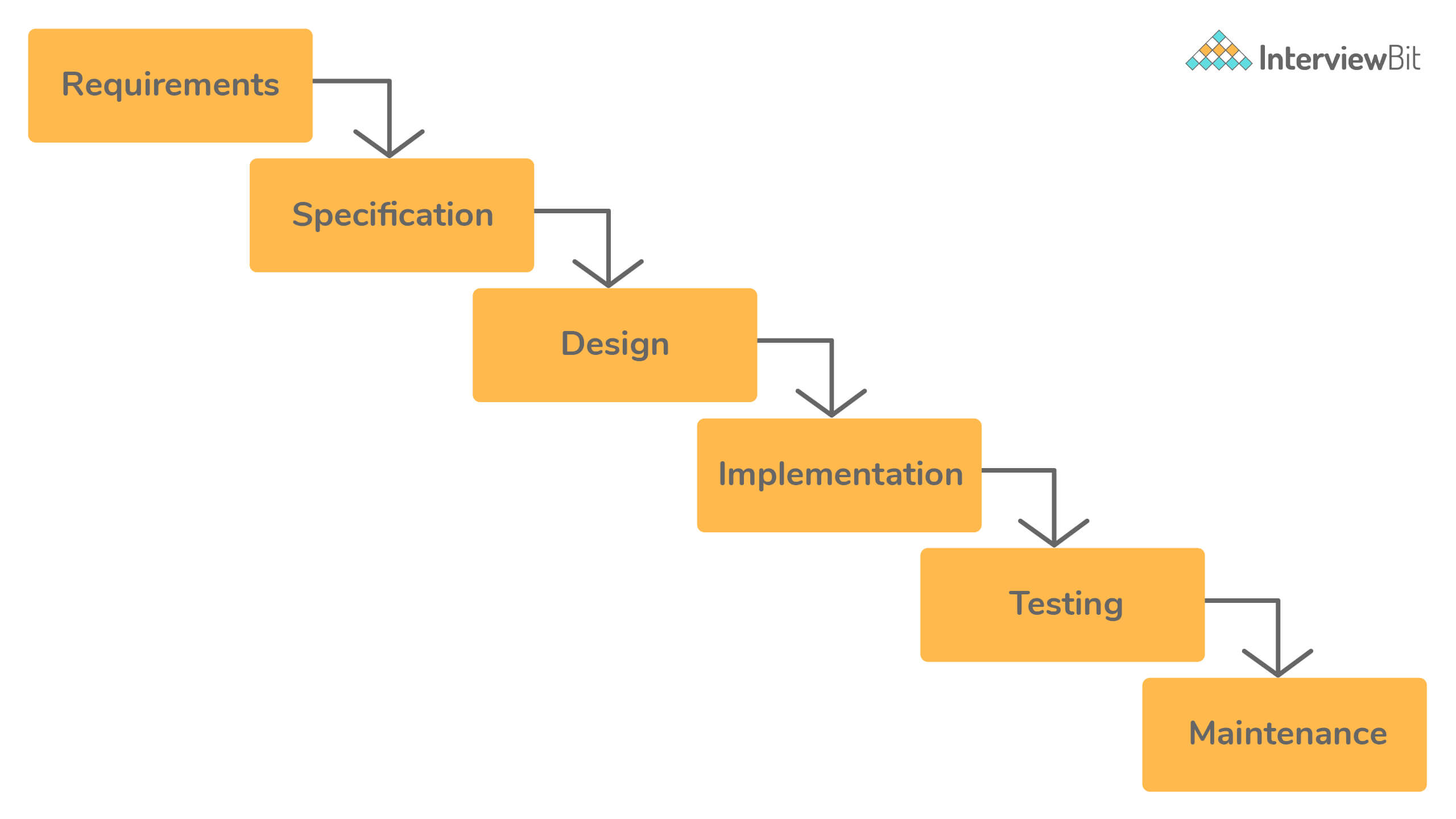
BVA (Boundary Value Analysis) is a black box software testing technique that uses boundary values to create test cases. Input values near the boundary have a higher probability of error, so BVA is used to test boundary values.

BVA includes values at the boundaries in the test cases. If the input falls within the boundary range, then the test is positive; if it falls outside, then it is negative. There are several types of values, including maximum or minimum, inside or outside edge, and typical or error values.

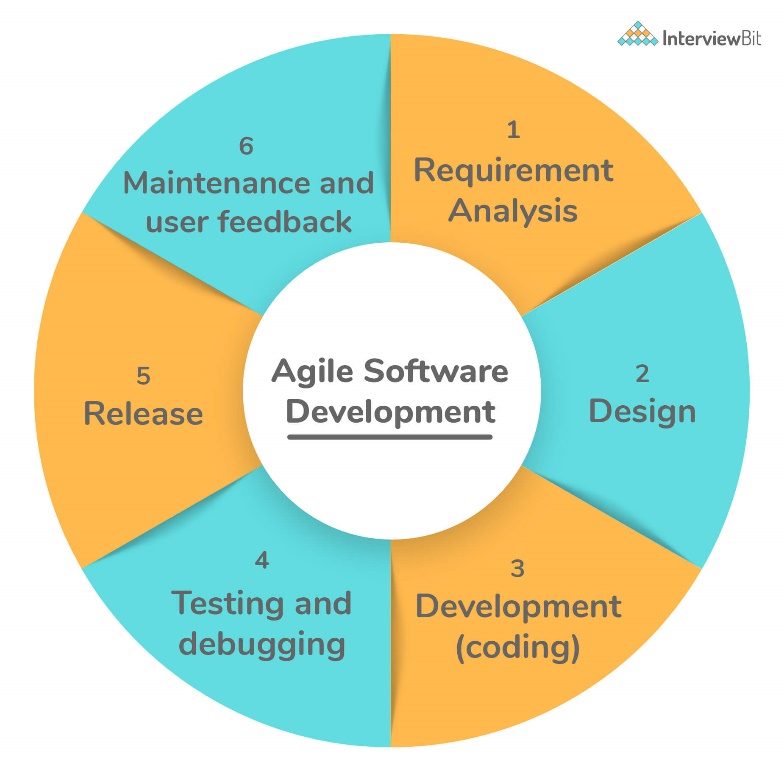
### **Role of testing in software development:**

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.



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.



### **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.

### **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.

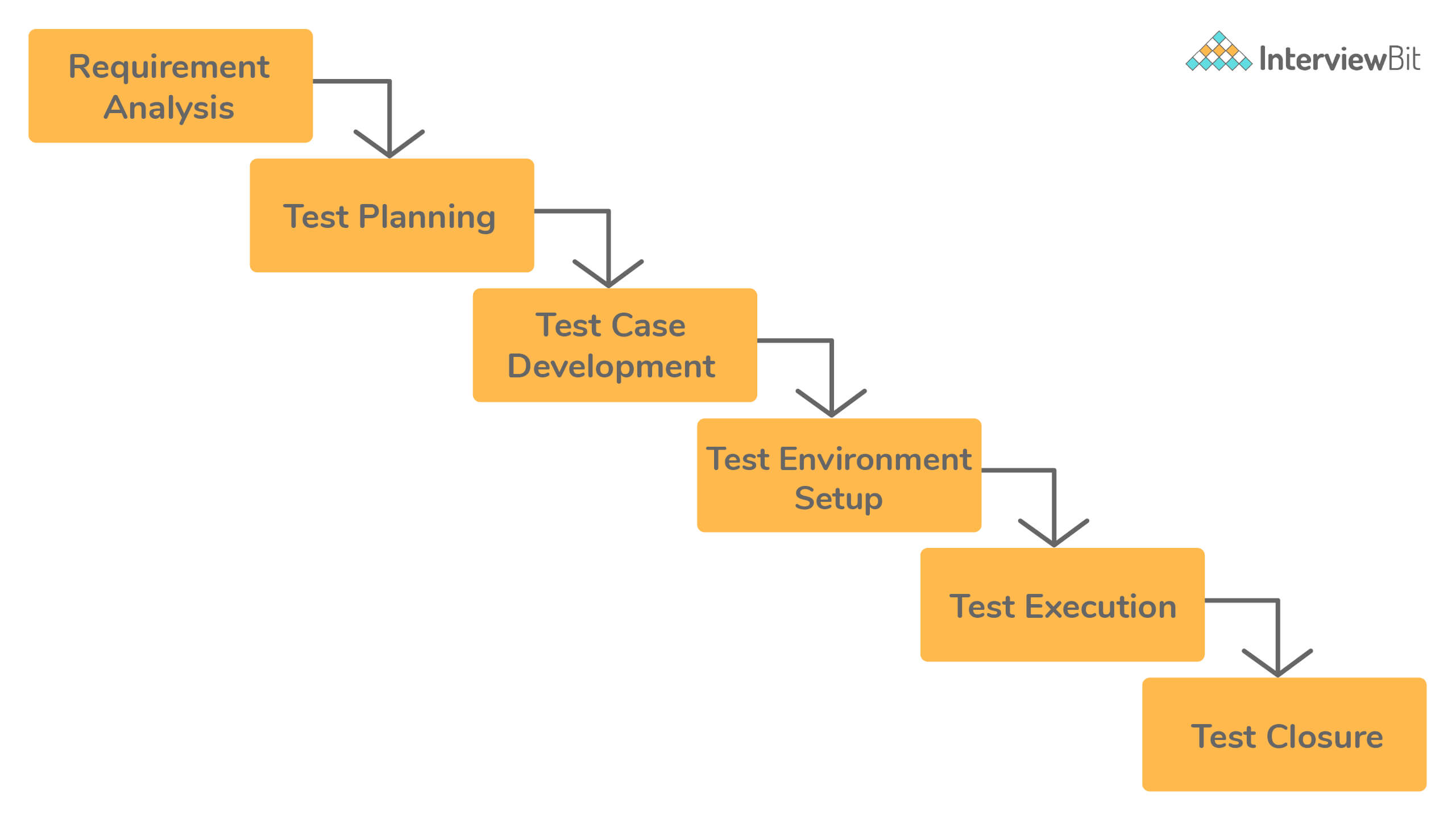
### **SDLC in software testing:**

In short, SDLC (Software Development Life Cycle) enables the development of high-quality, low-cost software with the shortest possible development time. A major objective of the SDLC is to produce high-quality software that meets and exceeds the expectations of customers.

SDLC provides a detailed plan with a series of stages, or phases, that encompass their own processes and deliverables. By adhering to the SDLC, developers can enhance the speed of their projects and minimize risks and costs.

### **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.

### **Functional testing:**

[**Functional testing**](https://www.interviewbit.com/functional-testing-interview-questions/) 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?

### **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.

### **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.

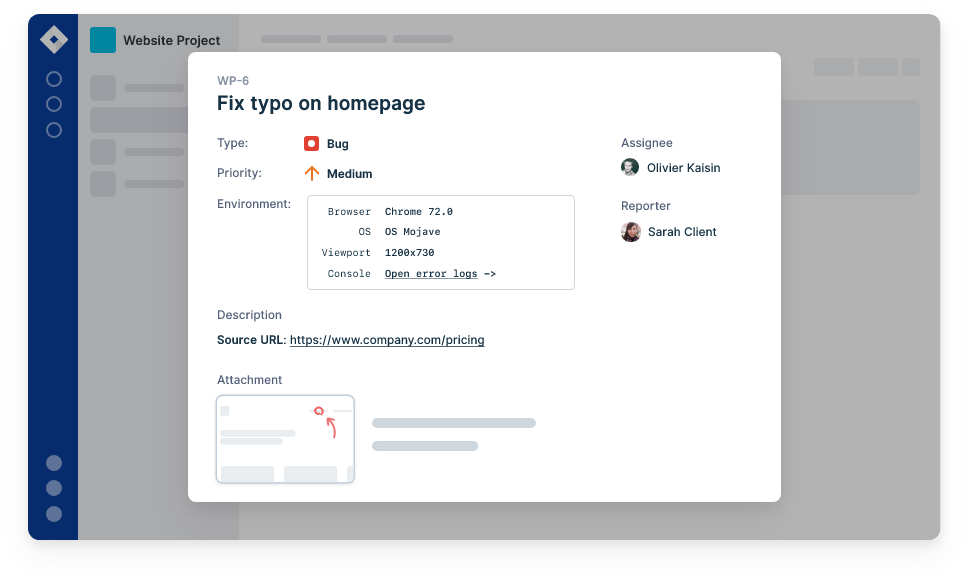
A detailed bug report is an important artifact produced during testing. It helps the team members with:

* 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. (TDVSSAR)

| **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. |
| **Resolution** | When a developer fixes the bug, they should include the cause for the bug and its resolution. It helps the team in the future when a similar bug resurfaces. |

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



### **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.

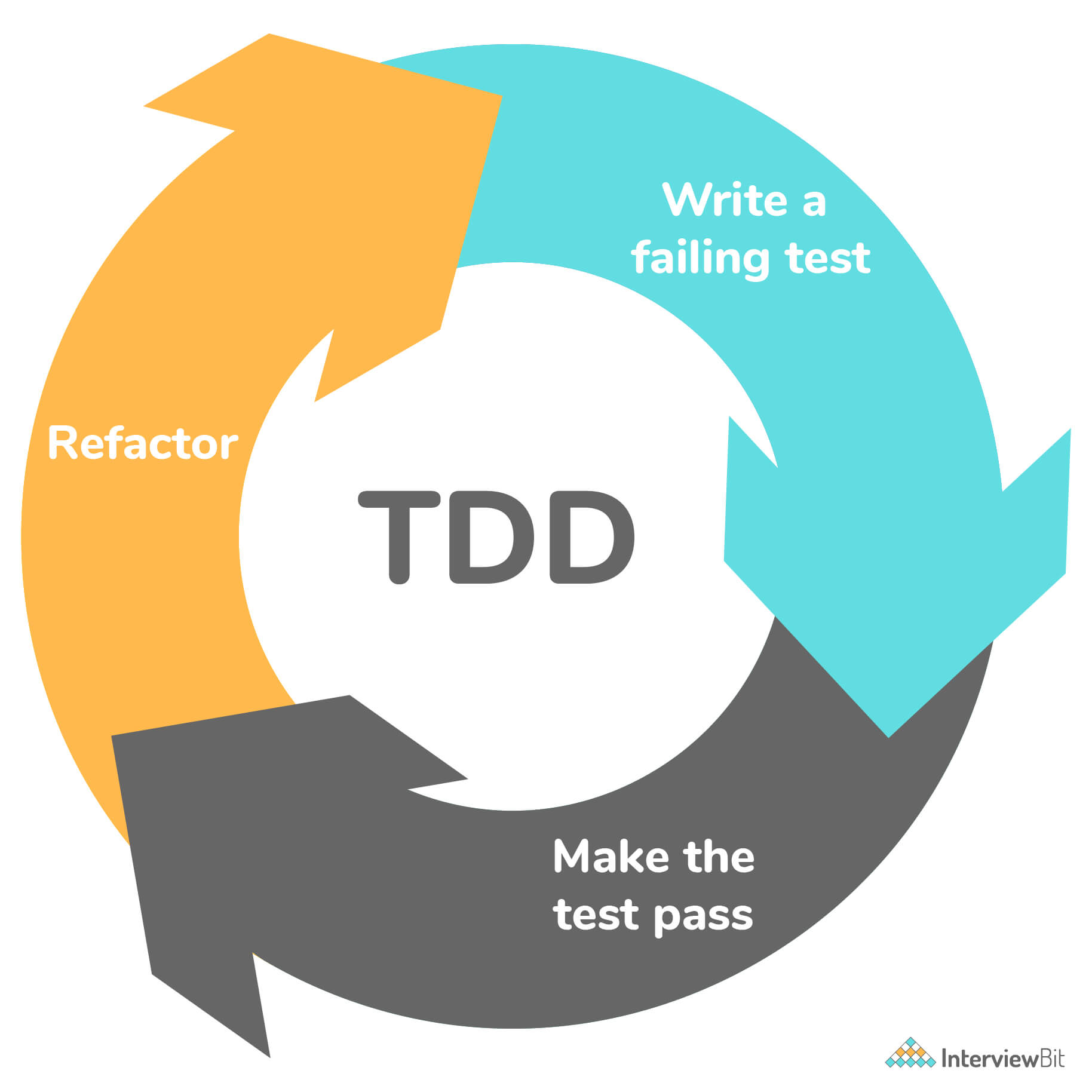
Here are some of the metrics derived from a record of the tests and failures:

* 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

### 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.

## **Selenium suite components**