Modual-2 (manual testing)

What is software testing

Software testing is the act of examining the artifacts and the behavior of the software under test by validation and verification. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but not necessarily limited to:

analyzing the product requirements for completeness and correctness in various contexts like industry perspective, business perspective, feasibility and viability of implementation, usability, performance, security, infrastructure considerations, etc.

reviewing the product architecture and the overall design of the product

working with product developers on improvement in coding techniques, design patterns, tests that can be written as part of code based on various techniques like boundary conditions, etc.

executing a program or application with the intent of examining behavior

reviewing the deployment infrastructure and associated scripts and automation

take part in production activities by using monitoring and observability techniques

Types of software testing

There are many types of software testing, but the two main categories are dynamic testing and [static testing](https://www.techtarget.com/whatis/definition/static-testing). Dynamic testing is an assessment that's conducted while the program is executed; static testing examines the program's code and associated [documentation](https://www.techtarget.com/searchsoftwarequality/definition/documentation). Dynamic and static methods are often used together.

Over the years, software testing has evolved considerably as companies have adopted Agile testing and Dev Ops work environments. This has introduced faster and more collaborative testing strategies to the sphere of software testing.

The following are the main types of software testing methodologies:

Integration testing. This groups together two or more [modules](https://www.techtarget.com/whatis/definition/module) of an application to ensure they function collectively. This type of testing also reveals interface, communication and data flow defects between modules.

Unit testing. Typically conducted during the application development phase, the purpose of unit testing is to ensure that each individual unit or component performs as expected. This is a type of [white box testing](https://www.techtarget.com/searchsoftwarequality/definition/white-box) and test automation tools -- such as Nun it, Unit and xUnit -- are typically used to execute these tests.

Functional testing. This entails checking functions against functional requirements. A common way to conduct functional testing is by using the [black box testing](https://www.techtarget.com/searchsoftwarequality/definition/black-box)

Security testing. This ensures the software is free of potential vulnerabilities, known flaws and security loopholes that might affect the user system and data. Security testing is generally conducted through [penetration testing](https://www.techtarget.com/searchsecurity/definition/penetration-testing).

Performance testing. This tests the performance and speed of an application under a given workload.

Regression testing. This verifies whether adding new features causes a decline in the functionality of an application.

Stress testing. This assesses the strength of software by testing how much load it can take before reaching a breaking point. This is a type of nonfunctional test.

Acceptance testing. This evaluates the entire system against the desired requirements and ensures the project is complete.

Installation testing

Most software systems have installation procedures that are needed before they can be used for their main purpose. Testing these procedures to achieve an installed software system that may be used is known as installation testing.

Compatibility testing

A common cause of software failure (real or perceived) is a lack of its [compatibility](https://en.wikipedia.org/wiki/Computer_compatibility) with other [application software](https://en.wikipedia.org/wiki/Application_software), [operating systems](https://en.wikipedia.org/wiki/Operating_system) (or operating system [versions](https://en.wikipedia.org/wiki/Software_versioning), old or new), or target environments that differ greatly from the original (such as a [terminal](https://en.wikipedia.org/wiki/Computer_terminal) or [GUI](https://en.wikipedia.org/wiki/GUI) application intended to be run on the [desktop](https://en.wikipedia.org/wiki/Desktop_metaphor) now being required to become a [Web application](https://en.wikipedia.org/wiki/Web_application), which must render in a [Web browser](https://en.wikipedia.org/wiki/Web_browser)). For example, in the case of a lack of [backward compatibility](https://en.wikipedia.org/wiki/Backward_compatibility), this can occur because the programmers develop and test software only on the latest version of the target environment, which not all users may be running. This results in the unintended consequence that the latest work may not function on earlier versions of the target environment, or on older hardware that earlier versions of the target environment were capable of using. Sometimes such issues can be fixed by proactively [abstracting](https://en.wikipedia.org/wiki/Abstraction_(computer_science)) operating system functionality into a separate program [module](https://en.wikipedia.org/wiki/Modular_programming) or [library](https://en.wikipedia.org/wiki/Library_(computing)).

Smoke and sanity testing

[Sanity testing](https://en.wikipedia.org/wiki/Sanity_testing) determines whether it is reasonable to proceed with further testing.

[Smoke testing](https://en.wikipedia.org/wiki/Smoke_testing_(software)) consists of minimal attempts to operate the software, designed to determine whether there are any basic problems that will prevent it from working at all. Such tests can be used as [build verification test](https://en.wikipedia.org/wiki/Build_verification_test).

Regression testing

Regression testing focuses on finding defects after a major code change has occurred. Specifically, it seeks to uncover [software regressions](https://en.wikipedia.org/wiki/Software_regression), as degraded or lost features, including old bugs that have come back. Such regressions occur whenever software functionality that was previously working correctly, stops working as intended. Typically, regressions occur as an [unintended consequence](https://en.wikipedia.org/wiki/Unintended_consequence) of program changes, when the newly developed part of the software collides with the previously existing code. Regression testing is typically the largest test effort in commercial software development,[[52]](https://en.wikipedia.org/wiki/Software_testing" \l "cite_note-52) due to checking numerous details in prior software features, and even new software can be developed while using some old test cases to test parts of the new design to ensure prior functionality is still supported.

Common methods of regression testing include re-running previous sets of test cases and checking whether previously fixed faults have re-emerged. The depth of testing depends on the phase in the release process and the [risk](https://en.wikipedia.org/wiki/Risk_management) of the added features. They can either be complete, for changes added late in the release or deemed to be risky, or be very shallow, consisting of positive tests on each feature, if the changes are early in the release or deemed to be of low risk. In regression testing, it is important to have strong assertions on the existing behavior. For this, it is possible to generate and add new assertions in existing test cases,[[53]](https://en.wikipedia.org/wiki/Software_testing" \l "cite_note-53) this is known as automatic test amplification.[[54]](https://en.wikipedia.org/wiki/Software_testing#cite_note-54)

Acceptance testing

Acceptance testing can mean one of two things:

A [smoke test](https://en.wikipedia.org/wiki/Smoke_testing_(software)) is used as a build acceptance test prior to further testing, e.g., before [integration](https://en.wikipedia.org/wiki/Integration_testing) or [regression](https://en.wikipedia.org/wiki/Regression_testing).

Acceptance testing performed by the customer, often in their lab environment on their own hardware, is known as [user acceptance testing](https://en.wikipedia.org/wiki/User_acceptance_testing) (UAT). Acceptance testing may be performed as part of the hand-off process between any two phases of development.[[citation needed](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed" \o "Wikipedia:Citation needed)]

Alpha testing

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing before the software goes to beta testing.[[55]](https://en.wikipedia.org/wiki/Software_testing#cite_note-55)

Beta testing

Beta testing comes after alpha testing and can be considered a form of external [user acceptance testing](https://en.wikipedia.org/wiki/User_acceptance_testing). Versions of the software, known as [beta versions](https://en.wikipedia.org/wiki/Beta_version), are released to a limited audience outside of the programming team known as beta testers. The software is released to groups of people so that further testing can ensure the product has few faults or [bugs](https://en.wikipedia.org/wiki/Computer_bug). Beta versions can be made available to the open public to increase the [feedback](https://en.wikipedia.org/wiki/Feedback#In_organizations) field to a maximal number of future users and to deliver value earlier, for an extended or even indefinite period of time ([perpetual beta](https://en.wikipedia.org/wiki/Perpetual_beta)).

• What is Exploratory Testing?

Exploratory software testing is done on the fly and relies on the tester to think beyond the limits of the scripted tests. Exploratory testers design a test, execute it immediately, observe the results, and use results to design the next test.  “emphasizes personal freedom and responsibility” in testing because the individual tester tests, learns and iterates throughout not looking for a pre-designed answer that cannot deviate. A lack of test scripts doesn’t mean a lack of preparation. Instead, it leads to less constraints on the tester. Exploratory tests are all about testers exploring an application to identify and document potential bugs. Testers embark on a process of investigation and discovery in order to effectively test a product. That's what exploratory testing is all about.

Learning

One of the most important functions that a tester needs is an understanding of the app or website that they are testing. This understanding provides context and includes information such as competitive benchmark data, industry knowledge and company details. For certain types of applications, a tutorial may be necessary. Much of this learning will happen in real-time, as they begin to explore and test the software.

An understanding like this ensures that the tester is able to take in all manner of inputs relating to the app or website when he or she is performing the actual test. Contextual knowledge allows testers to provide details surrounding results that they may find and whether or not they are applicable.

They also need to understand deeply the requirements of the specific test that was requested. A deep and comprehensive understanding of the testing requirements arms the tester with the right amount of knowledge to dive deep into the app or website. When the tester knows the requirements, including the scope of test coverage and the metrics they might need to consider, they can make sure to keep their testing and related bugs within those functional realms.

It is also important that the tester knows what the desired outcomes of the testing might be. It is not enough to merely look for bugs but to know what the end goal is for the testing cycle. For early iterations, they might be gathering test data on flaws. For later iterations, the goal may be to ensure that previous bugs have been resolved in such a way that new functional bugs have not been introduced (regression) somewhere else. Sometimes the software testing is to ensure that particular functions work as expected and no bugs found are the desired outcome per se.

Designing

A major difference between exploratory and scripted testing is in the test design. Designing differs from scripting in that while the test has specific parameters or rules, it is not done in a preset path or prescribed manner. Exploratory testers are able to conduct the test in a way which they deem fit and do not have a desired or expected outcome. Oftentimes, the exploratory testing technique leads to developing more rigorous, scripted test scenarios over time.

Designing also affords teams the option to map out the various techniques a tester might use. Test management can help decide the device, circumstances, or conditions if that has not been established yet by the test requester. Traditionally however, testers and test teams map out a time box of the test, the number of testers needed and other important pieces of the cycle. The cycle is still not a formal test case base nor is the tester writing out test cases during the testing itself. Testers can use notes, mind maps, flowcharts, decision tables or any manner of organizational tools at their disposal.

Executing

Finally, the tester is given the freedom to complete the test as they feel free to. As soon as the test idea is written or requested, it can be conducted. This freedom means that nobody is waiting for scripted requirements and creative work can begin. The tester can start observing and learning about the application or website. They’ll probe and explore how functions work, how they interact with one another and how those pieces work together so that further exploration can take place. Results are then compiled and reported back through the appropriate methods.

This method of test execution encourages and rewards both the product designed and tester by allowing an unconstructed approach to finding bugs. There is a stark difference between exploratory testing and abhoc testing although frequently the two are construed to mean the same thing. Adhoc approaches are distinguished by their lack of defined process and approaches.

When you think about this exploratory approach, it can be easy to think that the test methods rely on intuition or gut. However, this is not the case. Intuition and gut play a role during when a tester goes to test the application or software, but in the test plan there will be specific goals, functional areas and areas in which functionality is assessed.

It is best to imagine the functional exploratory tester as a detective who is investigating a crime. The crime has been committed by the software engineering team (expected functionality) and the detective is working his or her way through the clues to determine whether or not everything makes sense. Providing this level of freedom to assess your products and services ensures you deliver a level of quality your consumers absolutely demand.

What is traceability matrix

In [software development](https://en.wikipedia.org/wiki/Software_development), a traceability matrix (TM)[[1]](https://en.wikipedia.org/wiki/Traceability_matrix#cite_note-SST-1): 244  is a document, usually in the form of a table, used to assist in determining the completeness of a relationship by correlating any two [baseline documents](https://en.wikipedia.org/wiki/Baseline_(configuration_management)) using a many-to-many relationship comparison.[[1]](https://en.wikipedia.org/wiki/Traceability_matrix#cite_note-SST-1): 3–22  It is often used with high-level [requirements](https://en.wikipedia.org/wiki/Requirement) (these often consist of marketing requirements) and detailed requirements of the product to the matching parts of [high-level design](https://en.wikipedia.org/wiki/High-level_design), detailed design, [test plan](https://en.wikipedia.org/wiki/Test_plan), and [test cases](https://en.wikipedia.org/wiki/Test_case).

A requirements traceability matrix may be used to check if the current project requirements are being met, and to help in the creation of a [request for proposal](https://en.wikipedia.org/wiki/Request_for_proposal),[[2]](https://en.wikipedia.org/wiki/Traceability_matrix#cite_note-2) [software requirements specification](https://en.wikipedia.org/wiki/Software_requirements_specification),[[3]](https://en.wikipedia.org/wiki/Traceability_matrix#cite_note-3) various deliverable documents, and project plan tasks.

Common usage is to take the identifier for each of the items of one document and place them in the left column. The identifiers for the other document are placed across the top row. When an item in the left column is related to an item across the top, a mark is placed in the intersecting cell. The number of relationships are added up for each row and each column. This value indicates the mapping of the two items. Zero values indicate that no relationship exists. It must be determined if a relationship must be made. Large values imply that the relationship is too complex and should be simplified.

To ease the creation of traceability matrices, it is advisable to add the relationships to the source documents for both backward and forward traceability.[[5]](https://en.wikipedia.org/wiki/Traceability_matrix#cite_note-:0-5) That way, when an item is changed in one baseline document, it is easy to see what needs to be changed in the other.

* What is Boundary value testing

Boundary Value Testing is one of the popular software testing mechanism, where testing of data is done based on boundary values or between two opposite ends where the ends may be like from start to end, or lower to upper or from maximum to minimum. This testing process was introduced to select boundary values that came from the boundary based on the inputs at different ends of testing values. This black box testing strategy was introduced after equivalence class partitioning where the partition of classes takes place first followed by a partition at the boundaries.

What is Boundary Value Testing?

1. The testing is an analysis testing process that does testing practices at the boundaries of the partition covering all the testing defects where equivalence testing alone was difficult to handle those defects.
2. This testing machine is in conjunction with an equivalence testing mechanism that provides valid testing scenarios when there is ordered partition and the data consists of numeric format.
3. The boundary values of such a testing mechanism are identified by the values present at the extreme boundaries, i.e. minimum and maximum value.
4. This is used mainly to analyze the testing at the partition boundaries and also to detect anomalies that may occur during testing cases.
5. The black box testing techniques are helpful for detecting any errors or threats that happened at the boundary values of valid or invalid partitions rather than focusing on the center of the input data.
6. Boundary value analysis is a type of black box or specification based testing technique in which tests are performed using the boundary values.

Example:

An exam has a pass boundary at 50 percent, merit at 75 percent and distinction at 85 percent. The Valid Boundary values for this scenario will be as follows:

49, 50 - for pass

74, 75 - for merit

84, 85 - for distinction

Boundary values are validated against both the valid boundaries and invalid boundaries.

The Invalid Boundary Cases for the above example can be given as follows:

0 - for lower limit boundary value

101 - for upper limit boundary value

* What is Equivalence partitioning testing?

Equivalence Partitioning also called as equivalence class partitioning. It is abbreviated as ECP. It is a software testing technique that divides the input test data of the application under test into each partition at least once of equivalent data from which test cases can be derived.

An advantage of this approach is it reduces the time required for performing testing of a software due to less number of test cases.

Example:

The Below example best describes the equivalence class Partitioning

Assume that the application accepts an integer in the range 100 to 999

Valid Equivalence Class partition: 100 to 999 inclusive.

Equivalence partitioning is a black box test design technique in which test cases are designed to execute representatives from equivalence partitions. It can be applied to any level of the software testing, designed to divide a sets of test conditions into the groups or sets that can be considered the same i.e. system should handle them equivalently. Equivalence partitioning is also known as equivalence classes.

 Test cases for input box accepting alphabets from A to Z by using Equivalence Partitioning.

The test case should have all valid inputs for this partition. By this, we mean that pick nay alphabet between A to Z. If any other alphabet is chosen between these 26 alphabets the output will give us the same results. Hence we can conclude that one input is sufficient for testing this condition.

Input data other than these alphabets account as invalid input. These inputs can be either numerical values or special characters.

By this, you can categorize and segregate all possible test cases which can be divided into three classes. Values in test cases other than the selected ones from any class should give the same result. There is one representative selected from every input class which helps us in designing the test cases. The test case values are selected in such a way that the largest number of values should be tested and the same results should be received for all [test cases present in a](https://www.educba.com/test-cases-vs-test-scenario/) class, This can be either for valid values or invalid values.

What Is Integration Testing**?**

Integration testing is known as the second level of the [software testing process](https://www.simplilearn.com/tutorials/devops-tutorial/fundamentals-of-software-testing), following unit testing. Integration testing involves checking individual components or units of a software project to expose defects and problems to verify that they work together as designed.

As a rule, the usual software project consists of numerous software modules, many of them built by different[programmers.](https://www.simplilearn.com/how-to-become-programmer-article) Integration testing shows the team how well these disparate elements work together. After all, each unit may function perfectly on its own, but the pressing question is, “But can they be brought together and work smoothly?”

So, integration testing is the way we find out if the various parts of a software application can play well with others!

Incremental integration testing is performed by combining logically related two or more modules. Every module will be added one by one in the testing unit until the testers complete the whole system. With this approach, you can test the system for defects at an early stage in a smaller unit when it is reasonably easy to identify the cause. This type of testing intends to pass the feedback to the developers at the very start to fix the bugs. Bugs found with this testing can be fixed without disturbing the other modules. This method generally uses stubs and drivers to set up the transmission. Stubs and drivers are duplicate programs used to establish communication. You can perform this integration test in three different approaches.

Types of Integration Testing

the six most popular forms of integration testing.

Big Bang Method

This method involves integrating all the modules and components and testing them at once as a single unit. This method is also known as non-incremental integration testing.

This type of testing is usually performed only after all the modules are developed. Once developed, all modules will be coupled to form a single software system, and then the testing will be performed. This sort of testing generally suits smaller systems. Though every module will be developed before even starting the integration testing, the biggest disadvantage here is some of your resources will be unproductive as they have to wait for all the modules to be developed before starting the testing process thus, making it costly and time-consuming.

Bottom-Up Method

This method requires testing the lower-level modules first, which are then used to facilitate the higher module testing. The process continues until every top-level module is tested. Once all the lower-level modules are successfully tested and integrated, the next level of modules is formed.

In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested. The primary purpose of this integration testing is that each subsystem tests the interfaces among various modules making up the subsystem. This integration testing uses test drivers to drive and pass appropriate data to the lower level modules. Here the testing starts from the lowest module in the architecture. The testing control flow moves upwards from the bottom. This method will be executed whenever the top modules are under development. This method will use the drivers to restore the working of modules that are missing. This way of approach has a high success ratio and is an efficient way to test and develop a product. It is faster than the other traditional methods of testing.

Advantages:

In bottom-up testing, no stubs are required.

A principle advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.

Disadvantages:

Driver modules must be produced.

In this testing, the complexity that occurs when the system is made up of a large

number of small subsystems.

Bottom-Up Integration Testing – In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested. The primary purpose of this integration testing is that each subsystem tests the interfaces among various modules making up the subsystem. This integration testing uses test drivers to drive and pass appropriate data to the lower level modules.

Advantages:

In bottom-up testing, no stubs are required.

A principle advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.

Disadvantages:

Driver modules must be produced.

In this testing, the complexity that occurs when the system is made up of a large number of small subsystems.

3. Top-Down Integration Testing – Top-down integration testing technique is used in order to simulate the behaviour of the lower-level modules that are not yet integrated. In this integration testing, testing takes place from top to bottom. First, high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended. In this approach, testing is performed from the top-most module in the architecture. The testing control flow moves to the bottom from the top. This method will use stubs as duplicate programs to restore the working of modules that are missing. This method is comparatively easier than the bottom-up approach as it uses stubs, which are generally easier to write than the drivers. With this approach, you can find the interface errors with ease because of its incremental nature.

Advantages:

Separately debugged module.

Few or no drivers needed.

It is more stable and accurate at the aggregate level.

Disadvantages:

Needs many Stubs.

Modules at lower level are tested inadequately.

4. Mixed Integration Testing – A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level module have been coded and unit tested. In bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches.

Advantages:

Mixed approach is useful for very large projects having several sub projects.

This Sandwich approach overcomes this shortcoming of the top-down and bottom-up approaches.

5. Sandwich Integration Testing

It is a combination of Bottom-up and Top-down Approaches. In this approach, bottom modules are tested with top modules, at the same time, the top modules are tested with the lower modules. The goal here is to reach the mid module by testing both top and bottom modules simultaneously. This approach uses both stubs and drivers.

What determines the level of risk

Risk is the possibility of an event in the future, which has negative consequences. We need to plan for these negative consequences in advance so we can either eliminate the risk or reduce the impacts.

From the Testing perspective, a QA manager needs to be aware of these risks so he/she can minimize the impact on the quality of the software. Does this mean that the QA manager should address every risk that the project could face? In an ideal world, YES, but in all practicality, he would never have the time and resources to plan for every risk. Therefore we need to prioritize risks that would have severe consequences on software. How do we do that? We do that by determining the level of risk.

Dimensions of Risk

There are two dimensions of Risks that we should know.

Probability  -  Risk is always a possibility. The likelihood of risk is always between 0 % to 100 %. The probability can never be 0%; otherwise, risk will never occur. It can never be 100%; otherwise, it's not a risk; it is a certainty. E.g., We are hosting a website on a server that guarantees 99% uptime. What is the probability of the server going down? You guessed it right! It's 1 %.

Impact - Risk by its very nature has a negative impact. However, the size of the impact varies from one risk to another. We need to determine the impact on the project if the risk occurs. Continuing with the same example - What's the impact if the server goes down? Well, the site will not be accessible, so the impact is very high!

Levels of Risk

Based on these two dimensions, we determine the level of risk.

Level of Risk in Software = Probability of Risk Occurring  X  Impact if risk occurred

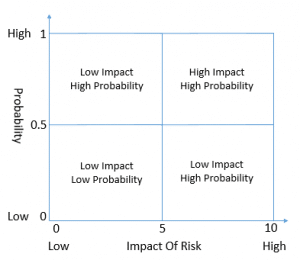
We can calculate the probability of risk between 0 - 1 with 0 depicting 0% occurrence and 1 depicting 100% occurrence. In this case, the classification of the impact is Low, Medium, and High. Some folks also classify it as Minimal, Minor, Moderate, Significant, and Severe. For the formula to calculate the level of risk, we can show the impact on a scale of 1 - 10 with 1 being the lowest impact and 10 being the highest impact. We can also use a range of 1-5, but irrespective of that, the core concept remains the same.

If we continue with our earlier example of server uptime of 99% - The Risk level will be calculated as :

Level of Risk = 0.1 X 10 = 1

0.1 is the probability of server going down (1% will translate to 0.1), and 10 is the impact on a scale of 1-10.

The Level of Risk calculation helps us in prioritizing risks. If we plot the probability and impact on a graph, we can classify the level of risk as below.



Let's understand these with examples - We will only discuss the concepts here. We will address the actual risk mitigation in our next article.

Low Impact Low Probability

These are the risks that have a low probability of occurrence, and the impact on software is pretty low.

E.g., Consider an e-commerce website that provides a Chat option so customers can chat with the service desk executive if they face any issues. Chat integration is a third-party plug-in, and it gives an uptime guarantee of 99%.

What's the probability of chat service not available? It's 1% which is pretty low

What's the impact if chat service goes down? The effect on software is pretty low, provided only a few users use chat service. Or there are other options to reach customer service executives (E.g., phone, and emails).

Such risks, which are low probability and the low impact, can be ignored. There isn't much value, add spending time on these risks.

Low Impact High Probability

These are the risks that have a high probability of occurrence, but the impact on software is pretty low.

E.g., We are migrating users from one website to another. The phone number format of both sites is different. As such, the probability of users losing their phone numbers in their profile is pretty high. However, As the phone number is not a mandatory field, it will not impact any user journeys. Also, a user can go ahead and update the phone number in the new format in My Account. Hence, the impact on software is low.

Such risks don't need much mitigation planning. These, however, need to be monitored to ensure that the impact remains low (What if the phone number becomes mandatory and user journey gets blocked ?)

High Impact Low Probability

These are the risks that have a low probability of occurrence, but the impact on software is pretty high.

Consider that we are hosting our test website on a server that guarantees 99.9 % uptime.

What's the probability of the server going down? It's 0.1 %, which is pretty low.

What's the impact if the server goes down? The website will not be accessible, and testing will completely stop. Do you see that it's a very high impact?

For such situations, we need to ensure that we have a mitigation plan if the risk does occur. It could be executing the tests in a different environment for the time the original server is down.

High Impact High Probability

These are the risks that have a high probability of occurrence, and the impact on software is pretty high as well.

Consider a situation where we are planning for testing software, and the timelines are very aggressive. The testing requires 10 Opium skill set resources; however, the availability of this skill set is very minimal in the organization.

What's the probability that we will not get the required resources on time? Well, It's pretty high, given that this skill set is scarce, and deployment of existing resources in on-going projects has finished already.

What's the impact if we don't get these resources in time? As the timelines are pretty aggressive, the impact on test completion will be pretty high!

High Impact and High Probability is the highest level of risk in software testing, and maximum planning and attention should go to this bucket. These risks have serious potential to derail testing thoroughly, and it could lead to delays in test completion or poor software quality.

For our current example, one mitigation could be to hire this skill set from the market. We can also hire contractors for a short duration to help in execution. As you would realize, the earlier we identify these risks, the easier it is to put the mitigation plan in place.

I hope you got a good understanding of risk in this tutorial. In our next tutorial, we will discuss product and project risks.

What is Alpha testing?

Alpha Testing is a type of software testing performed to identify bugs before releasing the product to real users or to the public. Alpha Testing is one of the user acceptance testings. This is referred to as alpha testing only because it is done early on, near the end of the development of the software. Alpha testing is commonly performed by homestead software engineers or quality assurance staff. It is the last testing stage before the software is released into the real world.

Objective of Alpha Testing:

The objective of alpha testing is to refine the software product by finding the bugs that were not discovered during the previous tests.

The objective of alpha testing is to refine the software product by fixing the bugs that were not discovered during the previous tests.

The objective of alpha testing is to involve customers deep into the process of development.

The objective of alpha testing is to give better insight into the software’s reliability at the early stages of development.

Alpha Testing Process:

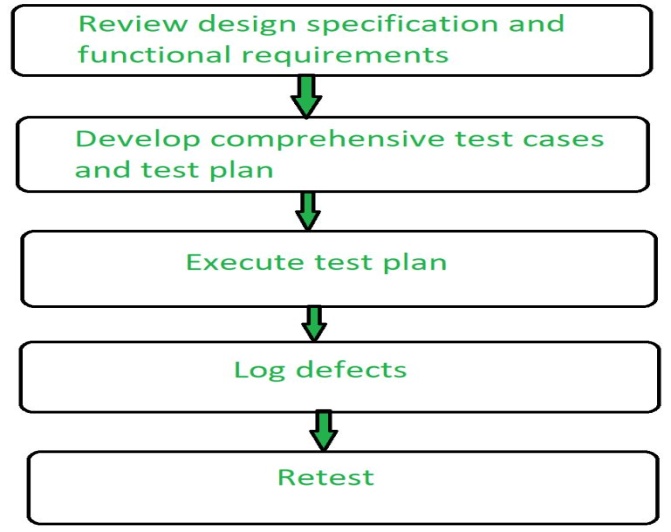
Review the design specification and functional requirements.

Develop comprehensive test cases and test plans.

Execute test plan

Log defects

Retest once the issues have been fixed

––--

Phases of Alpha Testing:

There are two phases in alpha testing:

1st Phase: The first phase of testing is done by in-house developers or software engineers. They either use hardware-aided debuggers or debugger software. The aim is to catch bugs quickly. Usually while alpha testing, a tester comes across to lots of bugs, crashes, missing features, and docs.

2nd Phase: The second phase of alpha testing is done by software quality assurance staff for additional testing in an environment. It includes a black box as well as white box testing.

Advantages of Alpha Testing:

Better insight about the software’s reliability at its early stages.

Free up your team for other projects.

It reduces delivery time to market.

Early feedback helps to improve software quality.

Disadvantages of Alpha Testing:

It will need a longer time for test plan execution if the project is large.

Sometimes, the defects in the products can be unknown during this alpha testing.

It is difficult to test the entire product since it is still under development.

For smaller projects, time spent on alpha testing is not worthy enough.

It does not carry out reliability and security testing.

This test will only cover the business requirements mentioned by the client. The project team will not go through the deep testing of each and every module.

It requires a separate lab environment for testing.

Alpha testing is the first end-to-end testing of a product to ensure it meets the business requirements and functions correctly. It is typically performed by internal employees and conducted in a lab/stage environment. An alpha test ensures the product really works and does everything it’s supposed to do. While “unit testing” and “smoke testing” of various individual components and features may have been conducted during the development process, an alpha test is an initial opportunity to evaluate the performance and comprehensive functionality of a given product release.

The primary [difference between an alpha test and a beta test](https://www.guru99.com/alpha-beta-testing-demystified.html) is who is doing the testing—alpha tests are typically performed by internal employees in a lab or stage environment, while [beta tests](https://www.productplan.com/glossary/beta-test/) are conducted by actual users in a production setting. The goal of the alpha test is to catch as many issues as possible before the product has any public exposure or usage.

Alpha tests can also be conducted using both “white box” and “black box” methods. In a white box setting, testers can “look inside” the product to see what’s happening during the testing, which is typically not possible in a production setting, while a black box test simply provides the inputs and confirms the outputs are returned as expected.

Because alpha testing occurs before the product is released, the white box view can provide additional insights to spot problems or troubleshoot bugs uncovered during the testing. Developers will typically immediately address issues discovered during alpha testing and update the test environment with fixes as soon as possible for additional testing.

Reliability and security testing are typically conducted during beta testing versus alpha testing because a stage/lab setting is not suitable for those environmental tests. The primary goal is to uncover “showstoppers” and other major bugs and issues as early as possible before moving forward with beta testing.

What is beta testing

Beta testing is an opportunity for real users to use a product in a production environment to uncover any bugs or issues before a general release.

Beta testing is the final round of testing before releasing a product to a wide audience. The objective is to uncover as many bugs or usability issues as possible in this controlled setting.

Beta testers are “real” users and conduct their testing in a production environment running on the same hardware, networks, etc., as the final release. This also means it’s the first chance for full security and reliability testing because those tests can’t be conducted in a lab or stage environment.

Beta tests can either be open or closed. In an open test, anyone can use the product and is usually presented with some messaging that the product is in beta and given a method for submitting feedback. In closed beta, the testing is limited to a specific set of testers, which may be composed of current customers, early adopters, and/or paid beta testers. Sometimes they are conducted by diverting a certain percentage of users to the beta site instead of the current release.

Testing can either last for a set period or run until new issues stop being reported and all-important ones have been addressed.

Planning: Like another testing process, beta testing also supports proper planning. In this stage, the team prepares a testing strategy and defines the goal of testing. In this case, the team establishes the need of users for testing, duration, and necessary details related to the process.

Participant Recruitment: This is the second stage of the beta process in which the team recruits a group of selected end-users for testing. This group can change as per the requirement of the organization and the product.

Product Launch: When a team of users (testers) recruited. The beta version of the product is launched or installed at the client or user side, and users will test the product for quality assurance.

Collect and Evaluate Feedback: When the testing finished, developers will collect the feedback provided by the testers and evaluate it. In the end, based on the feedback, issues, and bugs are fixed and resolved by the responsible individual team.

Closure: When all the problems fixed and the organization meets the exit criteria, beta testing achieved, and the rewards offered to the testing team.

Beta Testing is one of the Acceptance Testing types, which adds value to the product as the end-user (intended real user) validates the product for functionality, usability, reliability, and compatibility.

Inputs provided by the end-users help in enhancing the quality of the product further and lead to its success. This also helps in decision making to invest further in future products or the same product for improvisation.

Since Beta Testing happens at the end user’s side, it cannot be a controlled activity.

What is component Testing?

Component testing is defined as a software testing type, in which the testing is performed on each Another type of software testing is **Component Testing**. It is used to test all the components separately as well as the usability testing; interactive valuation is also done for each specific component. It is further known as **Module Testing or Program Testing and Unit Testing.**individual component separately without integrating with other components. It’s also referred to as Module Testing when it is viewed from an architecture perspective. Component Testing is also referred to as Unit Testing, Program Testing or Module Testing.

Generally, any software as a whole is made of several components. Component Level Testing deals with testing these components individually.

It’s one of most frequent black box testing types which is performed by QA Team.

As per the below diagram, there will be a test strategy and test plan for component testing. Where each and every part of the software or application is considered individually. For each of this component a[Test Scenario](https://www.guru99.com/test-scenario.html)will be defined, which will be further brought down into a High Level Test Cases -> Low Level detailed Test Cases with Prerequ

In order to implement the component testing, all the components or modules require to be in the individual state and manageable state. And all the related components of the software should be user-understandable.

This type of testing provides a way to finding defects, which happen in all the modules. And also helps in certifying the working of each component of the software.

Components testing is one of the most repeated types of black-box testing executed by the Quality Assurance Team.

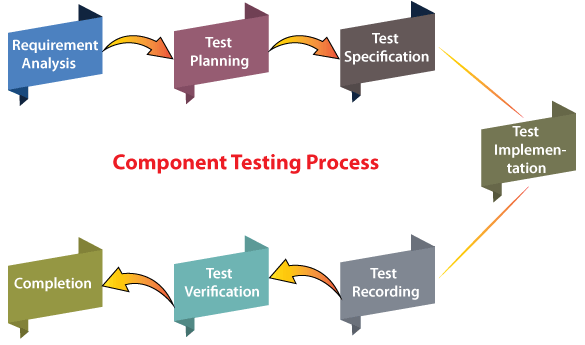
It can be performed individually, that is, in separation from the remaining system. However, it has relied on the model of the preferred life cycle.

The debugging or test structure tools can be used in the component testing.

In component testing bugs can be fixed as soon as possible when they are identified without keeping any records.

In simple words, we can say that the execution of components testing makes sure that all the application component is working correctly and according to the requirements. Component testing is executed before handing out with the integration testing.

The process of component testing can be completed in the following seven steps, as we can see in the below image:



Step1: Requirement Analysis

The first step of component testing is requirement analysis, where the user requirement associated with each component is detected.

Step2: Test Planning

Once the requirement analysis phase is done, we will mov to the next step of component testing process, which is test planning. In this phase, the test is designed to evaluate the requirement given by the users/clients.

Step3: Test Specification

Once the test planning phase is done, we will move to the next phase, known as test specification. Here, we will identify those test cases that needs to be executed and missed.

Step4: Test Implementation

The forth step in the component testing process is Test implementation. When the test cases are identified as per the user requirements or the specification, then only we can implement the test cases.

Step5: Test Recording

When all the above steps have been completed successfully, we will go to the next step that is, Test Recording. In this step of the component testing process, we have the records of those defects/bugs discovered during the implementation of component testing.

Step6: Test Verification

Once the bugs or defects have been recorded successfully, we will proceed to the test verification phase. It is the process of verifying whether the product fulfils the specification or not.

Step7: Completion

After completed all the above steps successfully, we will come to the last step of the component testing process. In this particular step, the results will be evaluated in order to deliver a good quality product.

When do we need to perform the Component testing?

When the unit testing is done on the particular application, we can proceed with the component testing. The components are tested once they are developed; thus, the output retrieved from a component under test depends on other components that are not created so far.

According to the development lifecycle model, component testing might be executed in segregation with other system components.

And the segregation is performed to stop the outside effects. Therefore, we will use Stubs and Drivers to pretend the interface between software components to test that component.

What is functional system testing

Functional testing is a type of testing that seeks to establish whether each application feature works as per the software requirements. Each function is compared to the corresponding requirement to ascertain whether its output is consistent with the end user’s expectations. The testing is done by providing sample inputs, capturing resulting outputs, and verifying that actual outputs are the same as expected outputs.

Some functional testing examples are:

Can users successfully log in to the application once they provide legitimate credentials?

Does the payment gateway reject the input and display an error message when a user keys in an invalid credit card number?

Do inputs to the “Add New Record” screen successfully add and save a new record to the database?

At the end of [functional testing](https://www.microfocus.com/solutions/functional-testing-software-testing), you should have software that has a coherent user interface, a consistent API, and seamlessly integrates with business processes.

Unite Testing :

is the first phase of software testing in the software development lifecycle (SDLC). This functional testing type is performed by developers and they write the scripts to validate whether the small units of the application are working as per the requirements or not.

Component Testing:

This functional testing type is much similar to unit testing, but the critical difference is that it is performed by testers and it tests each object of the application separately with or without isolation of other software objects.

Smoke Testing

[Smoke testing](https://www.testingxperts.com/blog/smoke-testing) is performed after each build is released. This functional testing type is also known as build verification testing. This process helps to test whether all the test environment aspects are running successfully and whether the build is stable or not and justifies if the build can be used for further testing process.

Sanity Testing:

Sanity testing is usually performed after smoke testing, i.e. after receiving a software build with minor changes made. In simpler words, it is used to validate the code changes introduced continue to work as expected.

Integration Testing:

Integration Testing is one of the important software testing method where individual modules of the application are combined and tested as a group to identify the functionality after combining different modules.

This functional [software testing method](https://www.testingxperts.com/blog/types-of-software-testing) helps in ensuring the new changes made to the code are not affecting the already existing features or functionalities of the application.

System Testing:

This is also a functional testing method that is performed after integration testing. This system testing phase is used for validating the fully integrated software application.

User Acceptance Testing (UAT):

This is the final software testing phase, wherein end-users or clients take up the roles of test engineers to validate whether the developed application is meeting the requirements or not. [UAT](https://www.testingxperts.com/blog/uat-testing) is an important functional testing type that is performed before moving the application for production.

Understand the requirements:

Before moving ahead for the functional testing process, the initial step is to have a thorough understanding of the business requirements. This is one of the very important steps to make further steps successful.

What is GUI Testing?

GUI testing is described as the testing of the application under the Test system graphical user interface. Graphical Testing consists of screen checks using controls such as icons, buttons, and menus with all types of the bar such as dialogues, menu bar, toolbar, Windows, etc. Basically, we can say that what we see is GUI. For a computer application, there are two kinds of interfaces. The command-line interface is the one that answers to the command text and the computer. On the other hand, GUI is a graphical user interface in which you use pictures rather than text to communicate with the computer.

Let’s take an example. If you or your friend visit a site which is Educba .com, First of all, he will see the home page, search button, course bar, and many more that are GUI (Graphical User Interface). A user sees no source code. The user can see the interface. The focus is particularly on the structure of the design, and images that work correctly or not. The connections are also accessible, and if you click, the button should operate. Also, neither image nor content must decrease or overlap if the user redesigns the screen. So this is a little bit of introduction regarding the GUI (Graphical User Interface) Testing.

Features of GUI

GUI Testing in Different Ways

Software experts use three major GUI testers worldwide to verify the precision and quality of graphical user interface elements.

Manual Based Testing

Record and replay

Model-Based Testing

1. Manual Based Testing

Testers manually verified all graphics for the company document with the prerequisites. The multiplication (33X5), for instance, can be checked by [manual testing](https://www.educba.com/manual-testing/).

2. Record and Replay

Record and Replay is an automated Graphical User Interface tool with all the test records at the time of testing.

3. Model-Based Testing

Model-based Testing acts as a graphic description. These tests predict the behaviour of the system, and this technique efficiently generates test cases. Decision and charts tables are some of the modelling techniques.

Check-List of GUI Testing

Check GUI elements like length, width, size, font, etc.

Check for the correct error message display.

Size of font and fonts readability.

Pictures should be aligned correctly.

The positioning for various resolutions of all GUI elements.

Advantages

By applying the GUI testing method during the early phases of the [SDLC](https://www.educba.com/what-is-sdlc/), the team will be able to take advantage of various benefits, such as accelerated growth, performance enhancement, and the risk decrease at the end of the cycle.

In addition, this testing offers other benefits such as:

Tests the interface from the point of view of customers.

The risk to the end of the development cycle is reduced efficiently.

Contributes to validating compliance with design specifications for the various icons and elements.

Improves product reliability and increases product quality.

Disadvantages

While there are numerous benefits of GUI testing, the tests still require acknowledgement. In addition, there are a few drawbacks. Here are, therefore, the drawbacks/disadvantages.

More memory resources are needed, which can slow the system.

The testing method takes time and may involve additional GUI software.

Given the frequent change in the implementation interface, the team could need to refactor a test script to enhance its precision.

The method of testing is difficult due to limited access or no access to the source code.

Examples

Test the height of the elements in size, location, width.

Testing for the displayed error messages.

Test the various parts of the screen.

Test the font whether or not it is readable.

We can also test the spelling.

Testing the screen in various sizes by zooming in and out such as 600×800, 640 x 480 etc.

Texts and other components such as buttons, icons and so on are in the correct location to test the alignment or not.

We can also test the front color.

Testing error messages colors and warning messages.

We can also test the clarity of the image

We can Also test the Alignment of the image.

What is Adhoc testing

Adhoc testing is frequently carried out to break the system in unusual ways. Adhoc testing's most remarkable aspect is that it lacks any test design approach for creating test cases.

The procedure is generally used to identify software flaws. Because adhoc testing lacks test cases, it is often carried out without documentation.

Take a closer look at the procedure. Ad-Hoc testing is a kind of testing that comes under the category of 'Unstructured Testing.'

Structured Testing Vs. Unstructured Testing

Structured Testing − Everything that happens throughout the testing method, from the development of test cases to their sequential execution, is written in this technique. This script is followed by the testers while doing tests.

Unstructured Testing − Testing is typically done by error guessing in this technique, where the testers construct the test cases throughout the testing process.

Adhoc Testing

Adhoc testing is a term that refers to testing that is done on the fly. It is a form of unstructured testing technique, as described above, in which no systematic strategy is made before the testing process begins. As a result, no requirement specification or test case preparation and design is done prior to testing.

Ad-hoc testing is generally carried out by a tester who is well-versed in the program under test, both in terms of what it does and how it works. This testing is carried out by constructing test cases at random using error guessing and running them without regard for any test criteria.

Finding possible regions of the program where mistakes may exist is an important aspect of this testing. Because of this, it's also known as Monkey Testing or Random Testing. As a result, only those testers with a thorough understanding of the product should execute this test.

Ad-Hoc testing has the benefit of saving time that would otherwise be spent on documents such as test requirements, test case planning, design, and so on. It is also often done after the structured testing has been completed. This is done in order to uncover software problems that would not have been found by following the previously prepared test cases.

Adhoc Testing Types

Following are some of the types of Adhoc Testing −

Buddy Testing

Monkey Testing

Pair Testing

Buddy Testing

In this sort of Ad-Hoc testing, at least two individuals work together to conduct the tests. At least one software tester and one software developer normally make up this team.

This sort of testing occurs after the unit testing of a module has been finished.

On that module, the two 'buddies' collaborate to develop legitimate test cases.

This is done to ensure that the tester does not report problems caused by incorrect test cases. This form of testing may also be thought of as a hybrid of unit and system testing.

Monkey Testing

The term 'monkey testing' refers to the unpredictability of the technique utilized in this testing.

Random inputs are presented to the program under test, and their associated outputs are monitored.

Any occurrences of errors, inconsistencies, or system failures are identified based on the acquired outputs.

Pair Testing

This kind of testing is similar to buddy testing. In this case, however, just a couple of testers collaborate on the modules for testing.

They collaborate to uncover faults and problems by sharing ideas, views, and expertise on the same computer.

To acquire a distinct perspective on each issue, testers are partnered according to their knowledge levels and experience.

Adhoc Testing Characteristics

This testing occurs after the program has been subjected to formal testing methodologies. The reason for this is because ad-hoc tests are performed to discover application abnormalities that cannot be foreseen prior to testing.

This testing can only be carried out by testers who have a deep understanding of how the program works. Because good 'error guessing' can only be done when the tester understands what the program does and how it operates, this is the case.

The Ad-hoc testing approach is best for detecting defects and inconsistencies in an application that lead to serious gaps. Errors like this are often difficult to spot.

This kind of testing requires less time than other types of testing. This is due to the fact that it is carried out without any previous planning, design, or structure.

Ad-hoc testing is done just once since any faults discovered need retesting.

Adhoc Tests Examples

Testing an application's functionality when the browser settings are altered. Identifying faults that occur when the JavaScript option is deactivated in various browsers, for example.

Putting the app through its paces on several platforms. It's critical to test if the produced application runs smoothly across a variety of operating systems and browsers.

Providing inputs to the system that are outside of the valid-inputs range in order to see whether the application's response is adequate.

Copying and modifying the application's URL to make it run in a different browser. This is done to ensure that unauthenticated access to the system is not granted to any unauthorized users.

Going through a series of random actions or traveling around the program at random in order to verify the outcomes achieved by using a certain combination of odd inputs.

What is white box testing and list the types of white box testing

The following article provides an outline for Types of White Box Testing. There are two popular box testing and they are white box testing and black box testing. We do testing to test all the standard of the require that have been developed. Like we will test functionality with different sort of inputs and analyzing whether outputs are proper. This testing is black box testing. We don’t consider the what is the box or requirement is made of. We just focus on functional part requirement that have been developed. In contrast to this we have white box testing which will test everything like standards of code as well. Any standard deviation is recorded as defect and needs to taken care. Thus white box testing is testing the internal logics, functions, conditions and in a way complete structure of the code.

Different Types of White Box Testing

White Box testing is also called as Structural Testing, Glass Testing, Open Box Testing or Clear Box Testing. We can clearly infer the names of white box testing is that internal structure is transparent and in way we will test the internal structure with the white Box testing.

The test case written as part of white box testing covers the code written, any branches, defined paths, internal logic and finally the statements of the code. Thus if we say that testing is going to perform white box testing then we may need to understand that he has some acquired prerequisite such as good knowledge of programming language in which the code is written, good understanding of programming of the logic, understanding of functional requirement of the project. If the tester is well equipped with this prerequisite then only he will be able to identify the chunk of code where there is malfunctioning. Hence most of the time developers only handle white box testing.

Given below are some of the popular types of white box testing:

Unit Testing

Static and Dynamic Analysis

Memory Leaks Testing

Statement Coverage

Branch Coverage

Mutation Testing

Security Testing

1. Unit Testing

It is most probably the first testing carried out for the application that is being developed. Unit testing as the name suggests will be carried out on a block of code or unit of code as it is being developed so that to test that particular functionality of unit is working fine. Most of the times unit testing is carried out by the developer only.

As an application developer, the developer when completes a part or functionality development then he carries out this unit testing to make sure it is working fine and depending on the result only they proceed further. Unit testing is a better way to identify the major bugs in the code. If ignored, then in the later part we may find it difficult to detect. The bugs identified are easy to detect, easy to fix and hence efficient on cost-wise as well.

2. Static and Dynamic Analysis

Static analysis testing is nothing but the following code to catch through any defects. Here we will not be executing the code. Dynamic analysis testing is running the code and analyzing the output to identify the possible defects. This type of testing is mainly for the validation of the data.

3. Memory Leaks Testing

If we work as IT professional then we might have heard optimization term quite many times. This is a major concern for any application and testing it thoroughly will help from saving log running time of applications. Generally, memory leaks are a major culprit for the code to run logger times. Hence any sort of memory leaks must be identified. This testing is mainly handled by QA specialist and this testing is must if you are handling large scale projects which have a heavy dependency on the memory.

4. Statement Coverage

The main purpose of this testing is to make sure that all the statements present in the code are executed and tested at least once. This makes sure that there are not any side effects or any functional defects with respect to the statements.

5. Branch Coverage

Generally, in projects with wider scope the code will not be written continuously. There will be branching out at some points to make sure that the code written till that point is working properly meeting all exceptions. Thus branching out is common in developing the code and branch coverage makes sure that all the branches are working as expected without any major drift from the requirements.

6. Mutation Testing

This testing is quite popular and this is used to test the code that has been modified after bug detection or corrections. This testing also helps in analysis of strategy like which will be the best strategy to adopt in enhancing and developing the functionality.

7. Security Testing

This testing is carried to identify the rigidness of application with respect to security and unauthorized access. It tests how the application performs when it comes to unauthorized and other security-related attacks such as hacking etc. This testing is also important as the organization will be intending to compromise on its product.

Advantages and Disadvantages of White Box Testing

Given below are the advantages and disadvantages mentioned:

Advantages

Below are the advantages:

The main advantage of white box testing is that it helps in optimizing the code.

This testing can identify hidden defects which may not be detected when we perform black box testing.

These testing can be easily Automated.

With the unit testing, we can say that the testing phase starts early in SDLC.

Disadvantages

Below are the disadvantages:

White Box testing will take a considerable amount of time since full structure and internal logic of the code tested.

This testing requires skilled and professional’s resources.

Some time white box testing can become quite complex.

What is black box testing? What are the different black box testing techniques

This article highlights the basics of black-box testing when to perform black-box testing and what are the benefits of using this form of the testing. It also gives insights into the various techniques used to perform black-box testing.

The black box is a powerful technique to check the application under test from the user’s perspective. Black box testing is used to test the system against external factors responsible for software failures. This testing approach focuses on the input that goes into the software, and the output that is produced. The testing team does not cover the inside details such as code, server logic, and development method.

Black box testing is based on the requirements and checks the system to validate against predefined requirements.

This article highlights the basics of black-box testing when to perform black-box testing and what are the benefits of using this form of the testing. It also gives insights into the various techniques used to perform black-box testing.

The black box is a powerful technique to check the application under test from the user’s perspective. Black box testing is used to test the system against external factors responsible for software failures. This testing approach focuses on the input that goes into the software, and the output that is produced. The testing team does not cover the inside details such as code, server logic, and development method.

Black box testing is based on the requirements and checks the system to validate against predefined requirements.

Types of Black Box Testing

There are many different types of Black Box Testing, some of them are given below:

Functional testing – This is a type of black box testing which is related to the functional requirements of a system; [Functional testing](https://reqtest.com/testing-blog/functional-vs-non-functional-testing/) is concerned only with the functional requirements of a system and covers how well the system executes its functions.

Non-functional testing – This black box testing type is not related to testing of specific functionality, Non functional testing is concerned with the non-functional requirements and is designed specifically to evaluate the readiness of a system according to the various criteria which are not covered by functional testing.

Regression testing – Regression Testing is performed after code fixes, upgrades or any other system maintenance to check the new changes has not affected any existing functionality.

Black box testing example:

A simple login screen of software or a web application will be tested for seamless user login. The login screen has two fields, username and password as an input and the output will be to enable access to the system.

A black box testing will not consider the specifications of the code, and it will test the valid username and password to login to the right account.

This form of testing technique will check the input and output.

A user logged in when inputs a present username and correct password

A user receives an error message when enters username and incorrect password

The black box testing is also known as an opaque, closed box, function-centric testing. It emphasizes on the behavior of the software. Black box testing checks scenarios where the system can break.

For example, a user might enter the password in the wrong format, and a user might not receive an error message on entering an incorrect password.

When we do Black Box testing?

Unlike traditional white box testing, black box testing is beneficial for testing software usability.

The overall functionality of the system under test

Black box testing gives you a broader picture of the software.

This testing approach sees an application from a user’s perspective.

To test the software as a whole system rather than different modules.

Various approaches to black-box testing

There are a set of approaches for black-box testing.

Manual [UI Testing](https://reqtest.com/testing-blog/gui-testing-tutorial/): In this approach, a tester checks the system as a user. Check and verify the user data, error messages.

Automated UI Testing: In this approach, user interaction with the system is recorded to find errors and glitches. Testers can set record demand as per schedule.

Documentation Testing: In this approach, a tester purely checks the input and output of the software. Testers consider what system should perform rather than how. It is a manual approach to testing.

What are Black Box testing techniques?

There are various [test case design techniques](https://reqtest.com/testing-blog/test-case-design-techniques/) applied for black-box testing:

Boundary Value Analysis

Equivalence partitioning

State Transition Testing

Decision Table Testing

Graph-Based Testing

Error Guessing Technique

1- Boundary Value Analysis

It is the widely used black-box testing, which is also the basis for equivalence testing. Boundary value analysis tests the software with test cases with extreme values of test data. BVA is used to identify the flaws or errors that arise due to the limits of input data.

For example: Taking inputs for a test case data for an age section should accept a valid data of anything between 1-100. According to BVP analysis, the software will be tested against four test data as -1, 1, 100, and 101 to check the system’s response using the boundary values.

2- Equivalence partitioning

This test case designing techniques checks the input and output by dividing the input into equivalent classes. The data must be tested at least once to ensure maximum test coverage of data. It is the exhaustive form of testing, which also reduces the redundancy of inputs.

For example: Taking inputs for a test case data for the example mentioned above will have three classes from which one data will be tested.

Valid class: 1 to 100 (any number), Invalid class: -1 (checking the lowest of lowest), Invalid class: 101(highest of highest).

3- State Transition Testing

This testing technique uses the inputs, outputs, and the state of the system during the testing phase. It checks the software against the sequence of transitions or events among the test data.

Based on the type of software that is tested, it checks for the behavioral changes of a system in a particular state or another state while maintaining the same inputs.

For example, A login page will let you input username and password until three attempts. Each incorrect password will be sent the user to the login page. After the third attempt, the user will be sent to an error page. This state transition method considers the various states of the system and the inputs to pass only the right sequence of the testing.

4- Decision Table Testing

This approach creates test cases based on various possibilities. It considers multiple test cases in a [decision table](https://reqtest.com/requirements-blog/a-guide-to-using-decision-tables/) format where each condition is checked and fulfilled, to pass the test and provide accurate output. It is preferred in case of various input combinations and multiple possibilities.

For example, A food delivery application will check various payment modes as input to place the order — decision making based on the table.

Case1: If the end-user has a card, then the system will not check for cash or coupon and will take action to place the order.

Case2: If the end-user has a coupon will not be checked for a card or cash and action will be taken.

Case3: if the end-user has cash, the action will be taken.

Case4: If the end-user doesn’t have anything, then action will not be taken.

5- Graph-Based Testing:

It is similar to a decision-based test case design approach where the relationship between links and input cases are considered.

6- Error Guessing Technique:

This method of designing test cases is about guessing the output and input to fix any errors that might be present in the system. It depends on the skills and judgment of the tester.

Comparison testing

This method uses the two different versions of the same software to compare and validate the results.

How to do Black Box testing?

When you get the basic understanding of black-box testing then the next question which comes up in mind is: How to perform the Black box testing? Below you can check the steps to perform this testing:

The first step to black-box testing is to understand the requirement specifications of the application under test. An accurate and precise [SRS document](https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document) should be there.

The next step is to evaluate the set of valid inputs and test scenarios to test the software.  The goal is to save time and [get good test coverage](https://reqtest.com/testing-blog/test-coverage-metrics/).

Prepare the test cases to cover a maximum range of inputs.

The test cases are run in the system to generate output, which is validated with the expected outcome to mark pass or fail.

The failed steps are marked and sent to the development team to fix them.

Retest the system using various testing techniques to verify its recurring nature or to pass it.

The black box testing can be easily used to check and validate the entire software development life cycle. It can be used at various stages such as unit, integration, acceptance, system, and regression to evaluate the product.

Mention what are the categories of defects

Software Defect is some kind of error, flaw or some kind of mistake from the development team which prevent the software from the smooth working. It directly affect software quality, software quality is some thing how smooth and reliable your software is. Smoothness and reliability is how less defects your software have.

Categories of defects:  
Categories of defects are: Errors of commissions, Errors of omissions, Errors of clarity, and Error of speed and capacity.

These are explained as following below.

1.Error of Commission:  
Commission means instruction or some kind of command given. Now the error in commission means the error in made in command or instruction. For example, suppose I wrote a loop which I was trying to run 10 times but I command it to run more than 10 times by mistake this is the error of commission.

2.Errors of Omissions:  
As name is already describing error of omission is some thing which happens accidentally. Omission word means something left out or executed. Practical most common example of this error is suppose we make a function in programming open its bracket but forget to close at the end.

3.Error of Clarity:  
The most common error in the natural languages. This error happens due to miss understanding between the developer and client. It travels most of the time from the requirements to the software.

4.Error of Speed or Capacity:  
The name of the error is itself enough i think to tell about it this error. Your software is working fine but not working in the required time this is the error of speed. When it comes to capacity it can be relevant to memory. For example, a small integer is declared where the long integer was required.

Mention what are the categories of defects

Imagine you’re manufacturing women’s blouses in Indonesia. A product inspection report from your QC partner shows about one third of the blouses have untrimmed threads.

Would you classify this quality defect as “minor” or “major”? You might say untrimmed threads are a “minor” defect that don’t affect the salability of the product. Or you might consider them a “major” defect if you distribute your product to a high-end retailer with a lower tolerance for quality issues.

[Defect classification](https://www.asiaqualityfocus.com/resources/glossary?__hstc=44765691.c4f37541df474b8370c34eb4602f77fc.1669634531468.1669634531468.1669634531468.1&__hssc=44765691.1.1669634531469&__hsfp=4244455505) is a vital step for determining if goods should pass or fail inspection. And considering the quantity and severity of different types of defects found helps you make an

informed shipping decision.

A professional inspection company often has established standards for classifying various types of defects for a particular product type. But it’s ultimately your responsibility as the buyer to decide your tolerance for different defects, often using a system like AQL (related: [The Importer's Guide to Managing Product Quality with AQL [eBook]](https://www.intouch-quality.com/about/inspection-level-and-selecting-an-aql-acceptable-quality-level-for-product-inspection/)).

3 TYPES OF DEFECTS EVERY IMPORTER NEEDS TO KNOW

Quality control professionals typically classify quality defects into three main categories: minor, major and critical. The nature and severity of a defect determines in which of the three categories it belongs.

Importers have the power to specify how many of each type of defect they’re willing to accept in their finished goods. This tolerance, in turn, impacts how many units per SKU an inspector would check during inspection—the sample size.

Importers commonly set these quality tolerances using a statistically valid acceptance sampling method known as acceptable quality limits, or acceptable quality levels (AQL)

Minor defects

Minor defects are usually small, insignificant issues that don’t affect the function or form of the item. In most cases, the customer wouldn’t even notice a minor defect on a product. And the customer wouldn’t likely return an item due to a minor defect alone.

Importers often set the highest tolerance—or AQL, if applying that standard—for minor defects in their inspected sample size. But an item can still fail inspection if the number of minor defects found exceeds the limit set by their tolerance

### Major defects

Major defects are more serious than minor defects. A product with a major defect departs significantly from the buyer’s product specifications.  Major defects are those which could adversely **affect the function, performance or appearance** of a product.

These defects are readily noticeable by the customer. And these defects would likely cause a customer to return the product, lodge a complaint or request a refund in response.

Most importers set a lower limit for major defects than minor defects in their inspected sample size. They’ll often accept an order with relatively few major defects. But they’re likely to reject an order, or ask their supplier to hold or rework it, if the goods fail inspection due to an excessive number of major defects found.

### Critical defects

Critical defects are the most serious of the three defect types. Critical defects render an item completely unusable and/or could cause harm to the user or someone in the vicinity of the product.

These defects put businesses at serious risk of product liability issues, lawsuits and product recalls.

Many importers have a “zero tolerance” policy for critical defects in their orders commensurate with this risk. An item will often fail product inspection if a single critical defect is found within the order.

EXAMPLES OF QUALITY DEFECTS IN DIFFERENT PRODUCTS

Different quality defects can appear in different products depending on materials, production processes and standards used. Below are some common examples of minor, major and critical defects in different product types.

Softlines

Softline products include raw fabric, home textiles, garments and other woven or knitted products and footwear. Garments, in particular, tend to be more vulnerable to quality defects because of the labor-intensive production processes involved in their manufacturing.

Human hands involved in sewing and stitching, for example, raise [variability](https://lmp.mit.edu/sites/default/files/documents/gershwin.pdf) in production. Whereas more automated production processes, like the use of injection molding in hardlines goods or robotics in electronic assembly, tend to lower variability. This typically results in more consistent product quality throughout a production run.

Some common softline defects include:

Minor defect: Untrimmed thread – these are a common quality issue in garment manufacturing. Factories can easily rework this defect by simply cutting the excess threads. You might also classify untrimmed threads as a major defect depending on your customers.

Major defect: Missing stitches – these typically appear due to a fault with the sewing machine or an operator error. Missing stitches impact the visual appearance of the product and can even affect seam strength, making this a more serious quality issue.

Critical defect: Needle found in item – needles can pose a hazard to the end user if they end up in the finished goods. This quality issue is almost always classified as a critical defect. And the presence of one such defect typically results in the item failing inspection.

Hardlines

Hardline products include a broad range of goods typically made of metal, wood or plastic materials. This category includes most furniture, sporting equipment, cookware, building materials and tools.

Some common hardline defects include:

Minor defect: Light abrasion on surface – an abrasion on the surface of a hardline item could be related to a production process or simply rough handling. Abrasions and other damage to the surface of an item typically won’t hurt the product’s salability, making this a minor defect.

Major defect: Deep scratch on item logo – damage to an item’s logo is often considered a major defect. Scratches on the logo can be difficult to repair. And they’re generally not tolerated in large number, as they can impact product salability and consumer perception of a brand.

Critical defect: Sharp point or burr on item – sharp points can cause harm to the end user and are often cause for failing an inspection. These hazards often lead to product recalls, which is partly why importers commonly classify them as critical defects.

is a broad term that can include fabricated steel, conduit piping, gas valves, wind turbines, maritime equipment and other machinery or materials designed for industrial use.

Some common defects in industrial products include:

Minor defect: Surface imperfections – a surface imperfection like a welding protrusion on a steel pipe typically won’t affect the use or functionality of an industrial product. But consider both the type of imperfection and the product’s intended use before classifying such a defect as minor.

Major defect: Non-critical dimensions out of tolerance – minor deviances in dimensions are undesirable in finished goods but often won’t impact the overall function of an industrial product. You might consider dimensional deviations more serious if they affect product function, performance or subsequent production processes.

Critical defect: Rust – corrosion before shipping can be a sign of accelerated degradation and product failure. Rust should be a serious concern for you if you’re importing gas or water pipes, for example.

CLASSIFYING DEFECTS IN QUALITY CONTROL CHECKLISTS

A QC professional can often suggest appropriate tolerances for known defects for your product. But it’s ultimately up to you as the buyer to specify your tolerance for each kind of quality defect.

Importers often list their defect classifications and tolerances in a document known as a quality control, or QC, checklist A QC checklist also typically includes other information like packaging requirements, on-site testing procedures and required inspection equipment.

This document should include an exhaustive list of common and known quality defects with your product type.

Accounting for every single quality defect may not be possible. But the more complete the list of potential defects you provide, the more likely your supplier will be to heed your tolerances. Your QC inspector is also far more likely to apply the same standard when checking your products against a complete QC checklist.

A defect classification list might look something like the below example for footwear:

This defect classification information can help:

Improve your supplier’s ability to proactively self-identify and correct quality defects before outside inspection

Ensure more accurate inspection results that match your quality tolerances and expectations

Reduce any cases of “[pending](https://www.intouch-quality.com/blog/product-inspection-pending-result)” results reported by the inspector due to unclear quality tolerances

This defect classification list might extend to include 20 or even 30 different types of defects depending on your product type. The more information you can provide, the better prepared your QC team and supplier will be.

Receive our latest news and make sure you don't miss anything!

Top of Form

Mention what bigbang testing is?

Big Bang Integration Testing is an integration testing strategy wherein all units are linked at once, resulting in a complete system. When this type of testing strategy is ad

opted, it is difficult to isolate any errors found, because attention is not paid to verifying the interfaces across individual units.

Disadvantages of Big-Bang Testing

Defects present at the interfaces of components are identified at very late stage as all components are integrated in one shot.

It is very difficult to isolate the defects found.

There is high probability of missing some critical defects, which might pop up in the production environment. Any defects in the interface of the modules are detected at a very late stage of software development.

It is challenging to isolate the defects, as the entire system is being tested at once, and any of the modules involved could have caused the error.

Big bang testing is also very time-consuming, as all possible test cases have to be tested at once.

It is very difficult to cover all the cases for integration testing without missing even a single scenario.

Big bang integration testing may be a process where all the software components are combined directly and make a sophisticated system. This unity of various modules is then tested as an entity. According to this checking method, the integration process will not be executed until all components are completed. Such an approach supports to save the tester’s time and effort while executing tests. To meet the desired goals and results, QA engineers have to record properly test cases and their outcomes.

In the world of software development, big bang testing is a type of integration testing.

Before software can be deployed and made available to users, it has to pass through a rigorous process in which all the different modules that make up the software are tested.

Testing ensures that the end product works exactly as it is intended to and does not run into any unforeseen errors.

In big bang integration testing, all the different modules are integrated simultaneously. After integration, the system is tested as a whole.

This is in contrast to the second model of integration testing, which uses an incremental approach

Advantages

Since all the modules are tested at once, big bang testing requires little to no planning beforehand.

All the modules are completed before testing begins.

Another advantage of using big bang testing is that it tests the entire system.

What is the purpose of exit criteria?

Exit criterion is used to determine whether a given test activity has been completed or NOT. Exit criteria can be defined for all of the test activities right from planning, specification and execution.

Exit criterion should be part of test plan and decided in the planning stage.

Examples of Exit Criteria:

Verify if All tests planned have been run.

Verify if the level of requirement coverage has been met.

Verify if there are NO Critical or high severity defects that are left outstanding.

Verify if all high risk areas are completely tested.

Verify if software development activities are completed within the projected cost.

Verify if software development activities are completed within the projected timelines.

The exit criteria get evaluated at the end of the testing cycle and is defined in Test Plan. It is the set of conditions or activities which must be fulfilled in order to conclude testing.

The Exit criteria define how much testing is enough and when testing activities can be declared complete. [Coverage](https://www.softwaretestinghelp.com/test-coverage/) and completion criteria are combined to define exit criteria for testing.

Often we see software testers very enthusiastic at the beginning of the project. We create [testing documents](https://www.softwaretestinghelp.com/why-documentation-is-important-in-software-testing/) such as Test Strategy, Test Plan or Test Cases eagerly and enthusiastically.

Then we get to testing software with a BANG! This is only amplified by the interesting defects we find at the beginning of the project. Getting them resolved will only add to our accomplishment.

As we find loads of defects and complete the first run we move on to the next phase. When we get to the second run we kind of relax and as is the general human tendency of [getting bored with testing the same thing](https://www.softwaretestinghelp.com/is-software-testing-boring-job/) in the second run.

Many testers feel that it becomes [monotonous work](https://www.forbes.com/sites/deborahljacobs/2014/04/04/how-to-deal-with-a-totally-boring-job/) in later runs and start losing interest in testing the same software over and over again. When we reach to, maybe the third run, one question starts haunting us and that is “When to Stop Testing the software?”

I bet you must have felt the same way and asked, “When to stop testing?”, at least once. I would say the question is “When, where and how to stop Testing?”

Conceptually we have read and many testers believe that there cannot be a specific condition or equation to decide “When to stop testing?” There are a number of factors to consider before we conclude on this question.

In today’s article, I would like to share my thoughts on how to conclude testing activities when we reach to a point in our testing cycle where we can say this testing is enough. We will do this with the help of a few real life examples in a typical testing cycle.

Decision to stop testing: Exit criteria

Let’s now try to understand – What are the most important factors to be considered while concluding testing activities? I feel the decision to stop testing is mostly dependent on Time, Budget and Extent of Testing.

The most common approach is to stop when either Time / Budget is exhausted or all test scenarios are executed. However, with this approach, we will be compromising on the quality of testing and this will not give enough confidence about the software.

What is Completion or Exit Criteria?

The exit criteria get evaluated at the end of the testing cycle and is defined in Test Plan. It is the set of conditions or activities which must be fulfilled in order to conclude testing.

The Exit criteria define how much testing is enough and when testing activities can be declared complete. [Coverage](https://www.softwaretestinghelp.com/test-coverage/) and completion criteria are combined to define exit criteria for testing.

What should be present in Exit Criteria?

Ideally, Exit or Stop Criteria is defined by combining various factors and hence is unique across all projects. It depends on the project requirement and hence should be defined during Test Planning; at the beginning of the project. Parameters defined in it should be quantified as much as possible.

Below are few pointers to be considered while defining Exit Criteria in case of Functional or System Testing. You may combine few or all the below factors while deciding where to stop testing as per your project needs.

When should "Regression Testing" be performed

Regression testing is in charge of identifying and preventing the aforementioned kinds of software regression. It makes sure that the previously developed and tested software still performs and behaves as intended after undergoing functional or non-functional changes. The term regression, in fact, means “a return to a former or less developed state.” The tests are performed to guarantee the new code hasn’t introduced new bugs or triggered any previously undiscovered ones that would cause your software to unintentionally regress.

Regression testing techniques

Depending on the complexity of your software solution, the size of your QA team, and the resources you’re blessed with, regression testing is performed using any combination of the following techniques:

Retest all involves rerunning all of your existing tests on the new codebase, retesting your entire software solution to reveal the regressions. This technique is used when the software undergoes significant functional and non-functional changes. QA teams usually perform it at the final stages of product delivery or before major releases. This practice is most suitable for small and very well designed projects while in a large project it can be very lengthy and resource-intensive. It is also quite infeasible and time-consuming to perform such a massive total of tests manually, thus, this technique usually implies [automated testing](https://testfort.com/automated-testing) using various tools and test-driven development practices.

Regression test selection is a less resource-intensive alternative to retest all. It takes a good strategy and a risk-based approach to perform regression test selection and yield good results. Using this technique, the QA team doesn’t write a whole new regression test suite but rather revises all the existing test suites. The team takes the affected by recent changes part of code and selects the most relevant test cases that have repeatedly found bugs. This technique takes a lot less time and effort to perform, can be effectively performed manually and helps the QA team discard obsolete test cases, which makes it a great choice for iterative development.

Test case prioritization is a technique that focuses on scheduling a limited set of test cases in a way that the test cases deemed most critical are executed ahead of the ones that have a smaller impact. It is used to increase a test suite’s rate of fault detection. There are two types of test case prioritization: general prioritization (targets all subsequent versions) and version-specific prioritization (targets a particular version).

The Hybrid technique is a combination of regression test selection and test case prioritization. Using this technique, QA teams first rerun the test cases of highest priority and then run all the remaining tests from the selected part of the test suite to make sure no obscure bug makes it through in-between versions.

What is 7 key principles? Explain in detail

It is important that you achieve optimum test results while conducting software testing without deviating from the goal. But how you determine that you are following the right strategy for testing? For that, you need to stick to some basic testing principles. Here are the common seven testing principles that are widely practiced in the software industry.

1) Exhaustive testing is not possible

Yes! Exhaustive testing is not possible. Instead, we need the optimal amount of testing based on the risk assessment of the application.

And the million dollar question is, how do you determine this risk?

To answer this let’s do an exercise

In your opinion, Which operation is most likely to cause your Operating system to fail?

I am sure most of you would have guessed, Opening 10 different application all at the same time.

So if you were testing this Operating system, you would realize that defects are likely to be found in multi-tasking activity and need to be tested thoroughly which brings us to our next principle[Defect](https://www.guru99.com/defect-management-process.html)Clustering

2) Defect Clustering

Defect Clustering which states that a small number of modules contain most of the defects detected. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.

By experience, you can identify such risky modules. But this approach has its own problems

If the same tests are repeated over and over again, eventually the same test cases will no longer find new bugs.

3) Pesticide Paradox

Repetitive use of the same pesticide mix to eradicate insects during farming will over time lead to the insects developing resistance to the pesticide Thereby ineffective of pesticides on insects. The same applies to software testing. If the same set of repetitive tests are conducted, the method will be useless for discovering new defects.

To overcome this, the test cases need to be regularly reviewed & revised, adding new & different test cases to help find more defects.

Testers cannot simply depend on existing test techniques. He must look out continually to improve the existing methods to make testing more effective. But even after all this sweat & hard work in testing, you can never claim your product is bug-free. To drive home this point, let’s see this video of the public launch of Windows 98

You think a company like MICROSOFT would not have tested their OS thoroughly & would risk their reputation just to see their OS crashing during its public launch!

4) Testing shows a presence of defects

Hence, testing principle states that –  Testing talks about the presence of defects and don’t talk about the

absence of defects. i.e. Software Testing reduces the probability of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.

But what if, you work extra hard, taking all precautions & make your software product 99% bug-free. And the software does not meet the needs & requirements of the clients.

This leads us to our next principle, which states that- Absence of Error

5) Absence of Error – fallacy

It is possible that software which is 99% bug-free is still unusable. This can be the case if the system is tested thoroughly for the wrong requirement. Software testing is not mere finding defects, but also to check that software addresses the business needs. The absence of Error is a Fallacy i.e. Finding and fixing defects does not help if the system build is unusable and does not fulfill the user’s needs & requirements.

To solve this problem, the next principle of testing states that Early Testing

6) Early Testing

Early Testing – Testing should start as early as possible in the Software Development Life Cycle. So that any defects in the requirements or design phase are captured in early stages. It is much cheaper to fix a Defect in the early stages of testing. But how early one should start testing? It is recommended that you start finding the bug the moment the requirements are defined. More on this principle in a later training tutorial.

7) Testing is context dependent

Testing is context dependent which basically means that the way you test an e-commerce site will be different from the way you test a commercial off the shelf application. All the developed software’s are not identical. You might use a different approach, methodologies, techniques, and types of testing depending upon the application type. For instance testing, any POS system at a retail store will be different than testing an ATM machine.

Difference between QA v/s QC v/s Tester

The terms quality assurance (QA) and quality control (QC) are often used interchangeably . Although both are tightly linked, are part of quality management, and sometimes difficult to differentiate, they are far from being the same. While [QA testing](https://synoptek.com/insights/it-blogs/best-practices-in-qa-testing-process/) focuses on providing assurance that quality requested will be achieved, QC testing focuses on fulfilling the quality requested. QA focuses on preventing defect while QC focuses on identifying the defect.

What is QA Testing?

QA testing is a function of software quality that assures that procedures and standards are appropriate for a project and are correctly executed. The QA team’s job is to improve development and test processes so that defects do not arise when the product is being developed.

What is QC Testing?

QC testing is a function of software quality that checks that the project follows standards, processes, and procedures laid down, and that the project produces the required internal and external deliverables. The QC team’s job is to identify defects after a product is developed but before it is released. [QC aims to identify (and correct)](http://www.business2community.com/tech-gadgets/quality-control-introduction-software-companies-01700269#M9UDtOJiWwRW6x2r.97) defects in the finished product.

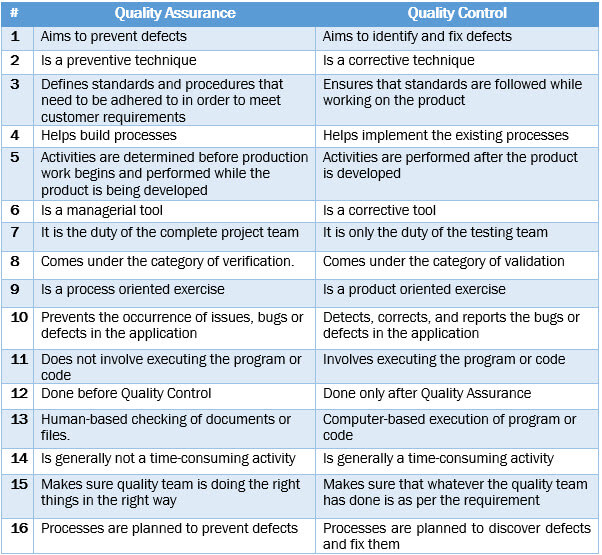
QA vs. QC: Comparison Overview

**Quality for Success**

One of the main reasons why many developers give [software testing](https://hackernoon.com/common-excuses-why-developers-dont-test-their-software-908a465e122c) a miss is because it increases development time; however, that’s not entirely true. Although software testing might seem challenging initially, it only aids in ensuring quality and reliability of your product.

Two certain ways of ensuring top-notch quality is to carry out QA and QC testing: [QA testing](https://www.mediapost.com/publications/article/303277/the-importance-of-quality-assurance.html) verifies that the processes used to manage and create deliverables are followed, and recognizes faults in the process, and QC testing ensures that the approaches, techniques, methods, and processes are followed in such a way that project deliverables meet the defined quality standards.

Therefore, software quality processes and delivering a quality product through QA and QC not only saves time and money, it mitigates risk and guarantees a competitive edge over rivals – a recipe for success!



Difference between Smoke and Sanity

Key Difference Between Sanity and Smoke Test

Smoke Testing has a goal to verify “stability” whereas Sanity Testing has a goal to verify “rationality”.

Smoke Testing is done by both developers or testers whereas Sanity Testing is done by testers.

Smoke Testing verifies the critical functionalities of the system whereas Sanity Testing verifies the new functionality like bug fixes.

Smoke testing is a subset of acceptance testing whereas Sanity testing is a subset of Regression Testing.

Smoke testing is documented or scripted whereas Sanity testing isn’t.

Smoke testing verifies the entire system from end to end whereas Sanity Testing verifies only a particular component.

What is Smoke Testing?

Smoke Testing is a software testing technique performed post software build to verify that the critical functionalities of software are working fine. It is executed before any detailed functional or regression tests are executed. The main purpose of smoke testing is to reject a software application with defects so that QA team does not waste time testing broken software application.

In [Smoke Testing](https://www.guru99.com/smoke-testing.html), the test cases chose to cover the most important functionality or component of the system. The objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system are working fine.  
For Example, a typical smoke test would be – Verify that the application launches successfully, Check that the GUI is responsive … etc.

What is Sanity Testing?

Sanity testing is a kind of Software Testing performed after receiving a software build, with minor changes in code, or functionality, to ascertain that the bugs have been fixed and no further issues are introduced due to these changes. The goal is to determine that the proposed functionality works roughly as expected. If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing.

The objective is “not” to verify thoroughly the new functionality but to determine that the developer has applied some rationality (sanity) while producing the software. For instance, if your scientific calculator gives the result of 2 + 2 =5! Then, there is no point testing the advanced functionalities like sin 30 + cos 50.

Difference between Smoke Testing and Sanity Testing

Following is the difference between Sanity and Smoke testing:

|  |  |
| --- | --- |
| Smoke Testing | Sanity Testing |
| Smoke Testing is performed to ascertain that the critical functionalities of the program is working fine | Sanity Testing is done to check the new functionality/bugs have been fixed |
| The objective of this testing is to verify the “stability” of the system in order to proceed with more rigorous testing | The objective of the testing is to verify the “rationality” of the system in order to proceed with more rigorous testing |
| This testing is performed by the developers or testers | Sanity testing in software testing is usually performed by testers |
| Smoke testing is usually documented or scripted | Sanity testing is usually not documented and is unscripted |
| Smoke testing is a subset of Acceptance testing | Sanity testing is a subset of [Regression Testing](https://www.guru99.com/regression-testing.html) |
| Smoke testing exercises the entire system from end to end | Sanity testing exercises only the particular component of the entire system |
| Smoke testing is like General Health Check Up | Sanity Testing is like specialized health check up |

Points to note about Smoke and Sanity Tests

Both Sanity and Smoke testing are ways to avoid wasting time and effort by quickly determining whether an application is too flawed to merit any rigorous testing.

Smoke Testing is also called tester [acceptance testing](https://www.guru99.com/user-acceptance-testing.html).

Smoke testing performed on a particular build is also known as a build verification test.

One of the best industry practice in software engineering, is to conduct a Daily build and smoke test in software projects.

Both smoke and sanity tests can be executed manually or using an [automation tool](https://www.guru99.com/automated-testing-tools.html).  When automated tools are used, the tests are often initiated by the same process that generates the build itself.

As per the needs of testing, you may have to execute both Sanity and Smoke Tests in the software build. In such cases, you will first execute Smoke tests and then go ahead with Sanity Testing. In industry, test cases for Sanity Testing are commonly combined with that for smoke tests, to speed up test execution. Hence, it’s a common that the terms are often confused and used interchangeably

Difference between verification and Validation

Verification process includes checking documents, design, code, and program, whereas Validation process includes testing and validation of the actual product.

Verification does not involve code execution, while Validation involves code execution.

Verification uses methods like reviews, walkthroughs, inspections, and desk-checking, whereas Validation uses methods like [black box testing](https://www.guru99.com/black-box-testing.html), white box testing, and non-functional testing.

Verification checks whether the software confirms a specification, whereas Validation checks whether the software meets the requirements and expectations.

Verification finds the bugs early in the development cycle, whereas Validation finds the bugs that verification can not catch.

Comparing validation and verification in software testing, the Verification process targets software architecture, design, database, etc., while the Validation process targets the actual software product.

Verification is done by the QA team, while Validation is done by the involvement of the testing team with the QA team.

Comparing Verification vs Validation testing, the Verification process comes before validation, whereas the Validation process comes after verification.

|  |  |
| --- | --- |
| Verification | Validation |
| The verifying process includes checking documents, design, code, and program | It is a dynamic mechanism of testing and validating the actual product |
| It does not involve executing the code | It always involves executing the code |
| Verification uses methods like reviews, walkthroughs, inspections, and desk- checking etc. | [It uses methods like Black Box Testing, White Box Testing, and non-functional testing](https://www.guru99.com/white-box-testing.html) |
| Whether the software conforms to specification is checked | It checks whether the software meets the requirements and expectations of a customer |
| It finds bugs early in the development cycle | It can find bugs that the verification process can not catch |
| Target is application and software architecture, specification, complete design, high level, and database design etc. | Target is an actual product |
| QA team does verification and make sure that the software is as per the requirement in the SRS document. | With the involvement of testing team validation is executed on software code. |
| It comes before validation | It comes after verification |

What is Validation in Software Testing?

Validation in Software Engineering is a dynamic mechanism of testing and validating if the software product actually meets the exact needs of the customer or not. The process helps to ensure that the software fulfills the desired use in an appropriate environment. The validation process involves activities like unit testing, integration testing, system testing and user acceptance testing.

Difference Between Verification and Validation in Software Testing

Here is the main difference between Verification and Validation in Software Testing:

Validation is done at the end of the development process and takes place after [verifications](https://tryqa.com/what-is-verification-in-software-testing-or-what-is-software-verification/) are completed.

It answers the question like: Am I building the right product?

Am I accessing the right data (in terms of the data required to satisfy the requirement).

It is a High level activity.

Performed after a work product is produced against established criteria ensuring that the product integrates correctly into the environment.

Determination of correctness of the final software product by a development project with respect to the user needs and requirements.

Explain types of Performance testing

Performance Testing is a software testing process used for testing the speed, response time, stability, reliability, scalability, and resource usage of a software application under a particular workload. The main purpose of performance testing is to identify and eliminate the performance bottlenecks in the software application. It is a subset of performance engineering and is also known as “Perf Testing”.

Performance testing will determine whether their software meets speed, scalability, and stability requirements under expected workloads. Applications sent to market with poor performance metrics due to nonexistent or poor performance testing are likely to gain a bad reputation and fail to meet expected sales goals.

Also, mission-critical applications like space launch programs or life-saving medical equipment should be performance tested to ensure that they run for a long period without deviations.

According to Dunn & Bradstreet, 59% of Fortune 500 companies experience an estimated 1.6 hours of downtime every week. Considering the average Fortune 500 company with a minimum of 10,000 employees is paying $56 per hour, the labor part of downtime costs for such an organization would be $896,000 weekly, translating into more than $46 million per year.

Types of Performance Testing

Load testing – checks the application’s ability to perform under anticipated user loads. The objective is to identify performance bottlenecks before the software application goes live.

Stress testing – involves testing an application under extreme workloads to see how it handles high traffic or data processing. The objective is to identify the breaking point of an application.

Endurance testing – is done to make sure the software can handle the expected load over a long period of time.

Spike testing – tests the software’s reaction to sudden large spikes in the load generated by users.

Volume testing – Under Volume Testing large no. of. Data is populated in a database, and the overall software system’s behavior is monitored. The objective is to check software application’s performance under varying database volumes.

Scalability testing – The objective of scalability testing is to determine the software application’s effectiveness in “scaling up” to support an increase in user load. It helps plan capacity addition to your software system.

## Common Performance Problems

Most performance problems revolve around speed, response time, load time, and poor scalability. Speed is often one of the most important attributes of an application. A slow-running application will lose potential users. Performance testing ensures an app runs fast enough to keep a user’s attention and interest. Take a look at the following list of common performance problems and notice how speed is a common factor in many of them:

Long Load time – Load time is normally the initial time it takes an application to start. This should generally be kept to a minimum. While some applications are impossible to make load in under a minute, Load time should be kept under a few seconds if possible.

Poor response time – Response time is the time it takes from when a user inputs data into the application until the application outputs a response to that input. Generally, this should be very quick. Again if a user has to wait too long, they lose interest.

Poor scalability – A software product suffers from poor scalability when it cannot handle the expected number of users or when it does not accommodate a wide enough range of users. [Load Testing](https://www.guru99.com/load-testing-tutorial.html) should be done to be certain the application can handle the anticipated number of users.

Bottlenecking – Bottlenecks are obstructions in a system that degrade overall system performance. Bottlenecking is when either coding errors or hardware issues cause a decrease in throughput under certain loads. Bottlenecking is often caused by one faulty section of code. The key to fixing a bottlenecking issue is finding the section of code causing the slow down and trying to fix it there. Bottlenecking is generally fixed by either fixing poor running processes or adding additional Hardware. Some common performance bottlenecks are

What is Error, Defect, Bug and failure

Testing is the process of identifying defects, where a defect is any variance between actual and expected results. “A mistake in coding is called Error, error found by tester is called Defect, defect accepted by development team then it is called Bug, build does not meet the requirements then it Is Failure.”

DEFECT:

It can be simply defined as a variance between expected and actual. The defect is an error found AFTER the application goes into production. It commonly refers to several troubles with the software products, with their external behavior or with its internal features. In other words, a Defect is a difference between expected and actual results in the context of testing. It is the deviation of the customer requirement.

Defect can be categorized into the following:

Wrong:

When requirements are implemented not in the right way. This defect is a variance from the given specification. It is Wrong!

Missing:

A requirement of the customer that was not fulfilled. This is a variance from the specifications, an indication that a specification was not implemented, or a requirement of the customer was not noted correctly.

Extra:

A requirement incorporated into the product that was not given by the end customer. This is always a variance from the specification, but maybe an attribute desired by the user of the product. However, it is considered a defect because it’s a variance from the existing requirements.

ERROR:

An error is a mistake, misconception, or misunderstanding on the part of a software developer. In the category of the developer, we include software engineers, programmers, analysts, and testers. For example, a developer may misunderstand a de-sign notation, or a programmer might type a variable name incorrectly – leads to an Error. It is the one that is generated because of the wrong login, loop or syntax. The error normally arises in software; it leads to a change in the functionality of the progra

BUG:

A bug is the result of a coding error. An Error found in the development environment before the product is shipped to the customer. A programming error that causes a program to work poorly, produce incorrect results or crash. An error in software or hardware that causes a program to malfunction. A bug is the terminology of Tester.

FAILURE:

A failure is the inability of a software system or component to perform its required functions within specified performance requirements. When a defect reaches the end customer it is called a Failure. During development, Failures are usually observed by testers.

FAULT:

An incorrect step, process or data definition in a computer program that causes the program to perform in an unintended or unanticipated manner. A fault is introduced into the software as the result of an error. It is an anomaly in the software that may cause it to behave incorrectly, and not according to its specification. It is the result of the error.

The software industry can still not agree on the definitions for all the above. In essence, if you use the term to mean one specific thing, it may not be understood to be that thing by your audience.

About 360logica

Backed by the team of dedicated and dynamic professionals with enriched experience,[360logica](https://www.360logica.com/), is a reputed offshore software testing company in India. With presence in premier locations in India and the United States of America, 360logica boasts of best-of-its-class infrastructure and skilled manpower along with hi-tech technology for testing and a wide base of satisfied clientele in different domains, including e-commerce, IT, healthcare, manufacturing and retail.

Difference between priority and severity

In the testing cycle of software, a bug usually acts as the most crucial entity. Two of the most important attributes that one can assign to a bug are Severity and Priority. Both of these attributes assist in fixing bugs pretty accurately and go through the process of release scheduling while still avoiding any major issues. Although they both serve a similar purpose, they still vary a lot. In this article, we will discuss the difference between severity and priority in testing. Read ahead to know about the individual roles that each of them plays.

What is Severity in Testing?

One can define Severity as the extent to which any given defect can affect/ impact a particular software. Severity is basically a parameter that denotes the impact of any defect and its implication on a software’s functionality. In other words, Severity defines the overall impact that any defect can have on a system.

For instance, consider if a web page or an application crashes after clicking on a remote link. In such a case, a user would rarely click on the remote link. Yet, the overall impact of an app crashing is very severe. Hence, the severity gets high, and yet the priority gets low.

What is Priority in Testing?

One can define Priority as a parameter for deciding the order in which one can fix the defect. In this, the defect with a higher priority first needs to get fixed. Priority basically defines the order in which one would resolve any given defect. The priority status defines if we should fix something or wait. The tester sets this priority status to the developer along with mentioning a time frame that can fix that defect. If they mention a higher priority, then the developer needs to fix it at the very earliest. Basically, the priority status comes into play according to the customer’s requirements.

For instance, let’s consider a case where one misspells the company name on a website’s home page. Thus, in this case, the Priority gets high while the Severity gets low for fixing it.

 d

 Discuss

 Practice

 Video

 Courses

Severity:  
Severity is defined as the extent to which a particular defect can create an impact on the software. Severity is a parameter to denote the implication and the impact of the defect on the functionality of the software.

Priority:  
Priority is defined as parameter that decides the order in which a defect should be fixed. Defect having the higher priority should be fixed first.

Difference between Severity and Priority in Testing:

| Severity | Priority |
| --- | --- |
| Severity is a parameter to denote the impact of a particular defect on the software. | Priority is a parameter to decide the order in which defects should be fixed. |
| Severity means how severe defect is affecting the functionality. | Priority means how fast defect has to be fixed. |
| Severity is related to the quality standard. | Priority is related to scheduling to resolve the problem. |
| Testing engineer decides the severity level of the defect. | Product manager decides the priorities of defects. |
| Its value is objective. | Its value is subjective. |
| Its value doesn’t change from time to time. | Its value changes from time to time. |
| Severity is of 5 types: Critical, Major, Moderate, Minor, and Cosmetic. | Priority is of 3 types: Low, Medium, and High. |

What is Bug Life Cycle

Stages of Bug Life Cycle in Testing A defect life cycle can consist of different stages depending on a company’s practices. Available defect management tools, number of team members, and adopted methodologies are among the things that can impact the workflow and, thus, a variety of bug life cycle stages. Below, you will find information about the frequently used statuses that designate different stages of the defect management process. Some of them are mandatory and some are optional, used to provide more details about the progress.

NEW Let’s say a software tester starts inspecting a product and finds a bug. What’s next? A specialist logs the defect into a tracking system, describing the steps for reproducing, environment details (OS, device, browser), and severity. The defect is assigned the New status.  
  
ASSIGNED A project manager, team lead, or developer in charge reviews the report. They check whether the bug is valid. If yes, they choose a developer to fix the defect. The choice might depend on the type of the bug (e.g., front-end or back-end, coded in PHP or Python, etc.) and available resources (who has enough time and expertise to solve the problem). The bug goes to the Assigned status. OPENED A developer who is to fix the issue sets the Opened status and starts analyzing the problem. A closer look at the defect may reveal some pitfalls. Thus, instead of fixing the bug, a developer has to change the status to one of the following: Duplicate — the identical problem was already reported.  
  
Rejected — the bug is not reproducible; a developer can’t understand the problem due to the ambiguous wording; there are no clear requirements; a QA engineer misunderstood the requirements; improper environment configuration during testing, etc. Deferred — the issue is not urgent and can be fixed later. Not a bug — the problem does not affect software functionality (e.g., a suggestion to change a font or background color). Cannot be fixed — for some reason, this bug has to become a feature, as there is no way to fix it (for now or at all). Need more info — a developer needs extra information about the defect to find its root cause. If the problem is reproducible and obvious, the developer starts working on fixing it. FIXED As soon as the developer makes changes to the code, the bug goes to Fixed. RETEST The task returns to the QA specialist, who checks if the problem is actually solved. REOPENED  
  
-

Explain the difference between Functional testing and Non Functional testing

Key Difference Between Functional Testing and Non Functional Testing

Functional testing verifies each function/feature of the software whereas Non Functional testing verifies non-functional aspects like performance, usability, reliability, etc.

Functional testing can be done manually whereas Non Functional testing is hard to perform manually.

Functional testing is based on customer’s requirements whereas Non Functional testing is based on customer’s expectations.

Functional testing has a goal to validate software actions whereas Non Functional testing has a goal to validate the performance of the software.

A Functional Testing example is to check the login functionality whereas a Non Functional testing example is to check the dashboard should load in 2 seconds.

Functional describes what the product does whereas Non Functional describes how the product works.

Functional testing is performed before the non-functional testing.

What is Functional Testing?

Functional testing is a type of testing which verifies that each function of the software application operates in conformance with the requirement specification. This testing mainly involves black box testing, and it is not concerned about the source code of the application.

Every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with the expected results. This testing involves checking of User Interface, APIs, Database, security, client/ server applications and functionality of the Application Under Test. The testing can be done either manually or using automation

What is Non-Functional Testing?

Non-functional testing is a type of testing to check non-functional aspects (performance, usability, reliability, etc.) of a software application. It is explicitly designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.

A good example of non-functional test would be to check how many people can simultaneously login into a software.

Non-functional testing is equally important as functional testing and affects client satisfaction

## What is functional testing?

Functional testing verifies that the operational execution of a program or mobile app happens according to the technical and business requirements. Only if every feature of a software system works correctly, it can pass a functional test.

Functional testing is usually conducted before non-functional testing and is done [manually](https://u-tor.com/topic/manual-testing). The tester provides specific inputs to the program and compares the result with the expected output. Functional testers are not concerned about the source code but focus on checking the functionality.

In layman terms, when you conduct functional testing you are interested in what the system can do. It is necessary to do this type of testing to ensure that the product does not have bugs or vulnerabilities.

## What is non-functional testing?

Non-functional testing checks all the aspects not covered in functional tests. It includes the performance, usability, scalability, and reliability of the software.

We conduct non-functional testing to make sure that the interests of the end-user are respected. The product will not enjoy success unless you manage to meet customer expectations. Non-functional testing is essential to add market value to the product.

This type of testing demands much more creativity from a tester. It has nothing in common with the mechanical work of clicking the right buttons. The testing specialist has to develop a strategy to collect customer expectations and offer a set of tests to check how these expectations are met.

The results of non-functional testing are measured on the scale. If functional testing specifies what the program should do, non-functional tests describe how it should behave. For example, how intuitive the interface is or how well the system operates under high load.

[Non-functional testing is often automated](https://u-tor.com/topic/automated-functional-testing-guide). Specialized tools help to model a real-life environment and see how the software is going to behave under pre-set conditions. Non-functional tests also rely on customer feedback and focus groups, especially in what concerns usability testing.

## Difference between functional and non-functional testing

The easiest way to understand the domains of functional and non-functional testing is to have a look at some questions they help to answer.

There are different [functional testing strategies](https://www.softwaretestinghelp.com/guide-to-functional-testing/) that you can use for your benefit. The best way to ensure functional coverage is to use both manual and automation testing methods.

Smoke testing. Before the actual testing, smoke testing is conducted to see that the main functionalities are working fine before fine-tuning other testing procedures.

Sanity testing. After a bug was fixed or functionality added, you need to understand that there are no issues that appeared due to this change.

Unit testing. Unit tests are done to check individual units or components of the system.

[Integration testing.](https://u-tor.com/topic/integration-testing) To test whether multiple software components function well together as a group, you will need integration testing.

Boundary value testing. When the software system sets limits on the input data, you need to check that this functionality is working correctly. Example: a password field that should include at least eight characters, one number, one uppercase letter, one lowercase letter, and one special symbol.

API testing. Integrating third-party services is always a potential source of bugs and memory leakages. If your project uses APIs, you should test that they are working correctly.

User acceptance. It validates that the functionality, accessibility, and quality are high enough for the application to be used in real-life.

[Regression testing](https://u-tor.com/topic/regression-testing-meaning). This one is done to verify that the recent changes or updates have no negative effect on the already existing functionality.

Globalization testing. Organizations today strive to design products that can perform successfully in international markets. This type of testing validates that the product will be accessible to users from foreign countries.

Interoperability testing. It confirms that the software can interact with other software systems and components without any compatibility issues.

Interface testing. To check whether the users will interact with the interface as expected, you need interface testing. To conduct it, you gather an end-user group that will perform specific tasks. It helps identify the elements of the software that the user often uses and whether these elements behave according to the requirements. It is essential not to confuse interface testing with usability testing. User interface testing checks that the interface works fine and passes the data to the system. Usability te

To create HLR & TestCase of

1)(Instagram , Facebook) only first page

|  |  |  |
| --- | --- | --- |
| Function id | Functional name | Description |
| 1 | [check https://web.whatsapp.com/](https://web.whatsapp.com/) | check it is opens the website |
| 2 | check snow dots button | checkif the snow dout button opens the sub main menu |
|  |  |  |
| 2.1 | check download button | check if click download button its opens google play store |
| 2.2 | check features button | check when click on features button it opens the page it give the whatsapp information |
| 2.3 | check privacy button | check if click on privacy button it opens discription of privacy which whatsapp provide to their customer |
| 2.4 | check help center button | check when click on help center it opens search bar their we can search vedio and discription which the user get |
| 2.5 | check get in touch button | check if click on get in touch button it opens contact whatsapp detail |
| 2.6 | check drop down button | check when user click drop down the button its opens types of language which are available on whatsapp |
| 2.7 | check cancel button | check if click on cancel button it's come again prvious page |
| 3 | check features button | check when click on features button it opens the page it give the whatsapp information |
| 4 | check download button | check if click download button its opens google play store |
| 5 | check whatsapp web | check when click on whatsapp web it opens website |
| 6 | check business button | check while click on business button its opens services about user business development |
| 7 | check privacy button | check if click on privacy button it opens discription of privacy which whatsapp provide to their customer |
| 8 | Check about button | Check while clicking on about buttons it opens whatsapp world wide discription |
| 9 | Check career button | Check while clicking on career button it open the job description |
| 10 | Check brand center button | Check while clicking on brand center button it provides brand overview |
| 11 | Check get in touch button | Check whiel ckicking on get in touch button it opens contect support from whatsapp |
| 12 | Check blog button | Chgeck while clicking on blog button it open which communities are available now on whatsapp for blog |
| 13 | Check whatsapp story button | Chgeck while clicking on whatsapp story button it oopens how do uyou grow your bussiness |
| 14 | Check mac/pc button | Check while clicking on mac/pc button it opens play store for download whatsapp application |
| 15 | Check android button | Check while clicking on android button it opens download link for android phones |
| 16 | Check iphone button | check while clicking on iphone button it opens app store for whatsapp link for iphoe os download |
| 17 | check help center button | check when click on help center it opens search bar their we can search vedio and discription which the user get |
| 18 | Check twitter button | Check while clicking on twitter button it opens twitter app |
| 19 | Check facebook button | Check while clicking on facebook button it opens facebook login page |
| 20 | Check coronavirus button | Check while clickingon corona virus button it opens guideline page about coronavirus |
| 21 | check privacy button | Check while clicking on privacy and terms button it opens whatspp legal page |
|  |  |  |

2) Instagram login page scenarios

|  |  |  |
| --- | --- | --- |
| Function id | Functional name | Description |
| 1 | Check https://www.instagram.com/ | Check if it is opens the website |
| 2 | Check login button | Check if the login button opens the userdashboard |
| 3 | Check login with in facebook | Check if the face book login button opens facebook homepage |
| 3.1 | Check login button | Check if the login button opens instagram 's home page /userdashboard |
| 3.2 | Check back arrow button | check it is closes the face book login page |
| 4 | check forgot password | check if it is opens trouble logging page |
| 4.1 | Check login link | check if it send you a link for get back your account |
| 4.2 | check creat new account button | Check it is open create account page |
| 4.3 | check sign up button | check whether click on sign up button it create new instagram account |
| 4.4 | Check back to login button | Check while click on back to loggin button we can see the login page |
| 5 | Check the footer get it on google play store button | Check while clicking on get on it google play store it opens play store and instagram app discription also |
| 6 | Check the footer grt it from microsoft | Check while clicking on get on grt it frim microsoft ot opens microsoft website |
|  |  |  |
|  |  |  |
|  |  |  |

3)face book scenarios for login page

|  |  |  |
| --- | --- | --- |
| Functional id | Functional name | Description |
| 1 | Check https://www.facebook.com/index.php | Check if it click on link it's opens facebook login page |
| 2 | Check login button | Check when user fill information and click on login button it's opens facebook userdashbord |
| 3 | Check forgotten password | check while user enter on forgotten password option it opens find your account page |
| 3.1 | check search button | Check if it is clicking on search button it will find our account |
| 3.2 | Check cancel button | check if clicking on cancel button it opens facebook login page again |
| 4 | check sign up for face book | check while click on sign up button it opens create new account page |
| 4.1 | check sign up button | check while clicking on sign up button its opens facebook home page |
| 4.2 | check already have account button | check while click on already have account button it opens facebook login page again |

What is the difference between the STLC (Software Testing Life Cycle) and SDLC (Software Development Life Cycle

STLC stands for Software Testing Life Cycle. STLC is a testing strategy performed by the testing team ensuring software quality standards are met. STLC is not to be confused with (SDLC) Software Development Life Cycle. SDLC embodies all software development phases while the software testing life cycle only focuses on the test phases.

Ivan Gekht, CEO of [Gehsoft](https://gehtsoftusa.com/" \t "_blank) concurs by stating, “STLC is a step-by-step approach to testing, where QA, throughout a sequence of predefined steps, ensures the quality of the product.” Casey Jordan CTO and Co-Founder of [easyDITA](https://easydita.com/" \t "_blank) explains, “The best organizations integrate testing with development in ways that reduce time to market, improve quality, and ensure features meet customer needs.”

This guide will explore the definition of software testing life cycle (STLC,) the differences between STLC & SDLC, hear from expert practitioners in the industry, and learn of automated software testing tools that foster productivity.

What is the difference between STLC vs SDLC?

There are differences between STLC and SDLC. The main difference is, Software Testing Life Cycle is a subset of the Software Development Life Cycle. Whereas, SDLC embodies the entire software creation process- from concept to completion. Testing software is a vital part of software creation. Therefore, STLC is a set of processes in the testable portions of creating software. “STLC is most concerned with how a feature works and less about how a feature is built,” adds Casey Jordan of easyDITA.

Andrei Mikhailau, Software Testing Director at [ScienceSoft](https://www.scnsoft.com/software-testing/experts/andrei-mikhailau" \t "_blank) elaborates, “As STLC is an inseparable part of SDLC aimed at software quality, each STLC stage should result in deliverables valuable for the SDLC process. STLC is aimed at ensuring the quality of developed software and its full compliance with the requirements specification. STLC comprises QA and testing practices (test planning, functional, integration, compatibility, performance, security, usability, regression test cases development & execution, and performing exploratory testing.)”

## STLC Phases

[Requirements Analysis](https://blog.autify.com/what-is-stlc-software-testing-life-cycle#RequirementsAnalysis)

[Test Planning](https://blog.autify.com/what-is-stlc-software-testing-life-cycle#TestPlanning)

[Test Case Development](https://blog.autify.com/what-is-stlc-software-testing-life-cycle#TestCaseDevelopment)

[Environment Setup](https://blog.autify.com/what-is-stlc-software-testing-life-cycle#EnvironmentSetup)

[Test Execution](https://blog.autify.com/what-is-stlc-software-testing-life-cycle#TestExecution)

[Test Cycle Closure](https://blog.autify.com/what-is-stlc-software-testing-life-cycle#TestCycleClosure)

Requirements Analysis

Naomi Bishop, CEO of [Surfky/Bidwise](https://www.bidwise.com/" \t "_blank), defines the requirements analysis phase as, “Analyze and research the criteria during this phase of the STLC. Participate in brainstorming sessions with other departments to determine if the criteria are testable. This stage aids in determining the extent of the research. If a function cannot be checked, let the team know during this process so that a mitigation plan can be formulated.” The QA team’s interaction with various key stakeholders is crucial at this stage.

Responsible QA team members collaborate with stakeholders. Any unclear aspects can be remedied here. We recommend the potential to automate testing capabilities. [Automation testing](https://blog.autify.com/top-6-test-automation-challenges?utm_source=blog&utm_medium=web&utm_campaign=us-content-marketing) tools such as Autify can be introduced. For example, we use our own test automation software to automatically test our product at this phase.

Test Planning in STLC

After the quality assurance team completes the requirements analysis stage they can move to the test planning phase. In this stage, a test plan document is created which acts as a strategic blueprint for testing. It details time and cost estimations, resources required, responsibilities delegation, and determining testing environments.

Naomi Bishop explains, “Test preparation is the first phase of the research process in most cases. We define the tasks and services that will aid in meeting the research goals during this process. We also aim to define the measures, as well as the way of collecting and monitoring those metrics, throughout the planning process.”

Test Case Development

[Test cases](https://blog.autify.com/how-to-write-test-cases?utm_source=blog&utm_medium=web&utm_campaign=us-content-marketing) are created during this phase. These tests check that every portion of the software works as intended for the end-user.

A test case includes test steps, testing data (such as login details,) an expected result upon successful execution, as well as unexpected results if there are failures. QA teams should strive for test case efficiencies as well. Traditionally, it has been known for QA teams to document test cases in spreadsheets, but this can be rather cumbersome especially at scale.

To reach 100% coverage, test scripts should be a mixture of manual and automated scenarios. The latter should dominate where possible. QA teams with the majority of manual testing experience bottlenecks, inefficient costs due to more time spent manually testing, and the potential for more bugs slipping into production due to the human error element.

Traditionally, test scripts are written in programming languages by skilled developers. This raises the barrier for entry, meaning, not everyone on the QA team can execute the tasks. Our software, [Autify](https://autify.com/?utm_source=blog&utm_medium=web&utm_campaign=us-content-marketing" \l "requestdemo" \t "_blank), makes testing easy for anyone on the team using our GUI rather than coding. Instead of writing test scripts in a programming language, our codeless platform writes the coded steps for users. Therefore, a tester would only need to focus on interaction with the browser. Engineers can focus on coding new features. Autify’s machine learning algorithms also handle maintenance too.

Environment Setup

Test environments are set up and deployed at this phase. This includes determining test data, hardware, and software implementation. Testing tools such as Autify, Selenium, or other [top software testing tools](https://blog.autify.com/10-best-software-testing-tools?utm_source=blog&utm_medium=web&utm_campaign=us-content-marketing) are used to aid in [automation testing](https://blog.autify.com/what-is-automation-testing?utm_source=blog&utm_medium=web&utm_campaign=us-content-marketing). It is vital that the testing environment closely emulates the software after deployment. For example, if a membership web application runs on the client’s server, then the testing environment should closely match. It’s common to perform smoke testing at this phase.

Test Execution

In the test execution phase, testers carry out the strategy outlined in the test plan, by executing test cases. The expected results are compared to actual results and reported back to the development team. Implementing sufficient bug tracking, defect reports, and test execution reports are key. When using an automation testing tool such as Autify, the AI automatically gathers reporting showing passed and failed test scenarios. It can also save mountains of time ([which are cost savings in software development](https://blog.autify.com/how-to-reduce-your-total-cost-of-ownership-tco-in-your-ui-test-with-codeless-automation?utm_source=blog&utm_medium=web&utm_campaign=us-content-marketing)) by detecting changes that may occur in the user interface.

Test Cycle Closure

The test cycle closure phase should conclude test execution. In essence, it’s a collection of test reporting results that should be presented, discussed, and analyzed among team members.

Naomi Bishop shares test cycle closure experiences from her team, ”[First,] verify that the evaluation has been done. If all test cases are executed or purposely mitigated. Verify that no defects of severity 1 have been opened.” She continues, ”[Second,] conduct a lesson-learned conference and prepare a lesson-learned study. (Include what went well, where changes should be made, and what should be improved.)”

Why Is Testing

z

|  |  |  |
| --- | --- | --- |
| Parameters | SDLC | STLC |
| Definition | SDLC, or software development life cycle, relates mainly to software development and includes all phases of software development, including testing. | In essence, STLC is related to software testing, meaning that it is a software testing process that entails several phases. |
| Relationship | As a whole, it covers the entire life cycle of the software and can be considered the predecessor. | Since it is part of SDLC and only involves testing, it is considered a child or successor. |
| Focus/Goal | SDLC aims to manage the entire process of software development from start to finish and to deliver a quality product that meets customer needs. | The focus is solely on test development and helps to make the testing process more sophisticated, consistent, and useful. |
| Performed | Phases of the SDLC are completed before those of the STLC. | Phases of STLC are carried out after the phases of SDLC. |
| Requirement Gathering | Business Analysts and Product Analysts collect requirements and prepare a Development Plan during the Requirements collection phase of the SDLC. | The QA (Quality Assurance) team will analyze requirement documents such as functional and non-functional requirements, and then prepare a System Test Plan as part of the Requirement Analysis phase of the STLC. |
| Different phases | The SDLC includes the following phases:Requirements Collection/PlanningAnalysis/DefiningDesigning the software programming or Coding (Building the Software)TestingDeployment/InstallationMaintenance | The STLC includes the following phases:Requirement AnalysisTest PlanningTest DevelopmentTest Environment SetupTest ExecutionTest Closure |
| Objective | Throughout the SDLC process, the intent is to overcome any hurdle on the way to successful software development. | Testing is only intended to find any weaknesses or pitfalls in the system. |
| Design phase | SDLC involves planning and designing software based on the requirements of the development team. | STLC involves the planning of tests by the testing team (Test Architect or Test Lead). |
| Coding phase | In this phase, programmers start writing code according to the designed document in any programming language to build the system from scratch. | Test cases and test scripts are developed by the testing team (quality assurance team) to verify the product’s quality. They prepare the test environment and execute the tests. |
| Environment Setup | As soon as the development team has written the code, they set up a test environment to validate it. | The testers ensure the test environment is prepared based on the prerequisites and conduct smoke tests to determine if the environment is stable enough for testing the product. |
| Testing Phase | The objective of this phase is to test the software. Among the activities included in this testing are Unit, Integration, System, Retest & Regression testing, etc., and the development team also participates in fixing reported bugs. | System integration testing is then conducted based on the test cases. All bugs and errors are reported, retested, and fixed. The product is also subject to regression tests and is signed off as soon as it meets the exit criteria. |
| Deployment/Product Release | As soon as the application has been approved by the various testing teams, it is deployed in a production environment for real end-users. | After the product has been deployed, smoke testing and sanity testing take place in the production environment, and the testing team prepares test reports and an analysis matrix for analyzing the product. |
| Maintenance | If necessary, post-deployment support, enhancement, and update are included. | The QA team runs regression tests to check deployed maintenance code. The team maintains test cases and automated scripts to make sure that tests are updated. |
| Members Required | Throughout the SDLC, more people (developers) are needed. | The QA team runs regression tests to check deployed maintenance code. The team maintains test cases and automated scripts to make sure that tests are updated. |
| Output | The end result of SDLC is the creation of reusable software systems. | STLC results in a tested software system. |

What is the difference between test scenarios, test cases, and test scrip

Test Scripts

This story begins with the most detailed way to document testing, the test script. When people talk about test scripts, they usually mean a line-by-line description of all the actions and data needed to perform a test. A script typically has ‘steps’ that try to fully describe how to use the program — which buttons to press, and in which order — to carry out a particular action in the program. These scripts also include specific results that are expected for each step, such as observing a change in the UI. An example step might be “Click the ‘X’ button,” with an example result of “The window closes.”

When a tester first starts a new job, they might not know much about the product, the business domain, or even software testing. Scripts can help bridge that gap. If the tester carefully follows the directions — enter the string ‘abc’, click the submit button, make sure the form submitted and the value was saved — the test idea will be covered enough to consider it ‘tested’.

There are a few drawbacks to consider before going all-in with detailed scripts. Active software projects change often — pages get redesigned, user experience changes, and new functionality is added. To be effective over time, testers have to make a continuous effort to update the scripts to match the new product. This can take time away from testing. Another drawback is that scripted tests are often designed to test one specific thing repeatedly, using the same steps and the same data each time the test is executed. This means that if there are bugs that lie outside the directions given in the test script, they will not be found unless the tester strays from the script. Scripted tests do not always encourage testers to use the creativity and technical skill required to find hidden bugs.

Test Cases

The second most detailed way of documenting testing work is to use test cases. Test cases describe a specific idea that is to be tested, without detailing the exact steps to be taken or data to be used. For example, a test case might say “Test that discount codes can be applied on top of a sale price.” This doesn’t mention how to apply the code or whether there are multiple ways to apply the code. The actual testing that will cover this test case may vary from time to time. Will the tester use a link to apply a discount, or enter a code, or have a customer service rep apply the discount, or will they feel compelled to test every way to add a discount that they can think of? Test cases give flexibility to the tester to decide exactly how they want to complete the test.

This flexibility from test cases is both good and bad. Flexibility is beneficial when the tester is familiar with testing and familiar with the software under test and the current set of risks in the software. If the tester clearly understands what has already been tested, what has changed recently in the program, and how users typically use the program, they will choose an approach in their testing that will exercise both the most common user paths, and the less common paths that are most likely to reveal bugs.

On the other hand, if the tester does not have a good understanding of how the program is used, the recent risks to the program, and how to evaluate those risks as a tester, they may not have the information or skill they need to assess the actions required to reveal important bugs.

Test Scenarios

The least detailed type of documentation is the test scenario. A test scenario is a description of an objective a user might face when using the program. An example might be “Test that the user can successfully log out by closing the program.” Typically, a test scenario will require testing in a few different ways to ensure the scenario has been satisfactorily covered. Just based on that light description, the tester might choose to close the program through the menu option, kill it through the task manager, turn the computer off, or see what happens when the program runs out of memory and crashes. Since test scenarios offer little information about how to complete the testing, they offer the maximum amount of flexibility to the tester responsible for them.

Like test cases, the flexibility that comes with using test scenarios creates similar benefits and drawbacks. Testing skill and domain knowledge make it easier for the tester to break test scenarios down into the relevant test ideas, select the approach that makes most sense, and perform tests that find important problems. This work is fun and challenging for a skilled tester, but it may be difficult or impossible for a novice unless they are able to collaborate with others to get the needed skill and perspective.

Explain what Test Plan is? What is the information that should be covered

Test Plan

A Test Plan is a detailed document that describes the test strategy, objectives, schedule, estimation, deliverables, and resources required to perform testing for a software product. Test Plan helps us determine the effort needed to validate the quality of the application under test. The test plan serves as a blueprint to conduct software testing activities as a defined process, which is minutely monitored and controlled by the test manager.

As per ISTQB definition: “Test Plan is A document describing the scope, approach, resources, and schedule of intended test activities.”

Let’s start with following Test Plan example/scenario: In a meeting, you want to discuss the Test Plan with the team members, but they are not interested – .

What is the Importance of Test Plan?

Making Test Plan document has multiple benefits

Help people outside the test team such as developers, business managers, customers understand the details of testing.

Test Plan guides our thinking. It is like a rule book, which needs to be followed.

Important aspects like test estimation, test scope,[Test Strategy](https://www.guru99.com/how-to-create-test-strategy-document.html)are documented in Test Plan, so it can be reviewed by Management Team and re-used for other projects.

How to write a Test Plan

You already know that making a Test Plan is the most important task of Test Management Process. Follow the seven steps below to create a test plan as per IEEE 829

Analyze the product

Design the Test Strategy

Define the Test Objectives

Define Test Criteria

Resource Planning

Plan Test Environment

Schedule & Estimation

Determine Test Deliverables

Step 1) Analyze the product

How can you test a product without any information about it? The answer is Impossible. You must learn a product thoroughly before testing it.

The product under test is Guru99 banking website. You should research clients and the end users to know their needs and expectations from the application

Step 2) Develop Test Strategy

Test Strategy is a critical step in making a Test Plan in Software Testing. A Test Strategy document, is a high-level document, which is usually developed by Test Manager. This document defines:

The project’s testing objectives and the means to achieve them

Determines testing effort and costs

Step 2.1) Define Scope of Testing

Before the start of any test activity, scope of the testing should be known. You must think hard about it.

The components of the system to be tested (hardware, software, middleware, etc.) are defined as “in scope“

The components of the system that will not be tested also need to be clearly defined as being “out of scope.”

Defining the scope of your testing project is very important for all stakeholders. A precise scope helps you

Give everyone a confidence & accurate information of the testing you are doing

All project members will have a clear understanding about what is tested and what is not

Step 2.2) Identify Testing Type

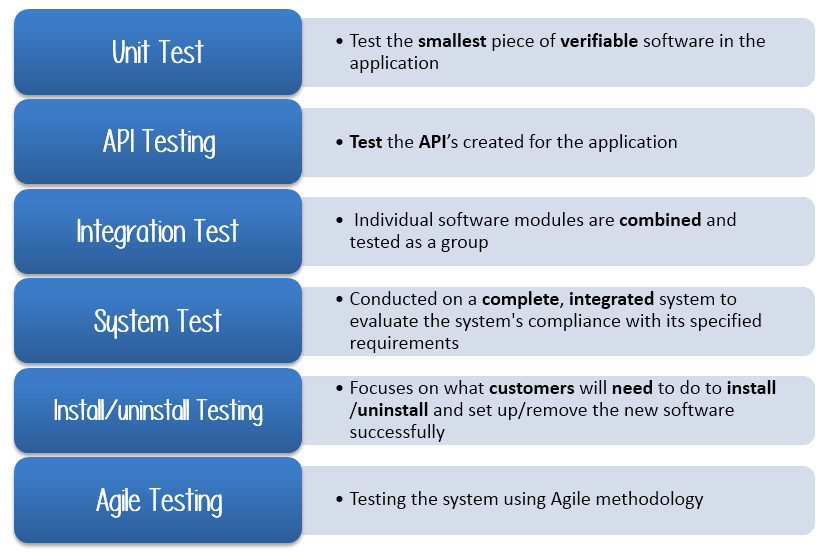
A Testing Type is a standard test procedure that gives an expected test outcome.

Each testing type is formulated to identify a specific type of product bugs. But, all Testing Types are aimed at achieving one common goal “Early detection of all the defects before releasing the product to the customer

Step 2.3) Document Risk & Issues

Risk is future’s uncertain event with a probability of occurrence and a potential for loss. When the risk actually happens, it becomes the ‘issue’.

In the article [Risk Analysis and Solution](https://www.guru99.com/how-precaution-becomes-cure-risk-analysis-and-solutions-in-test-management.html), you have already learned about the ‘Risk’ analysis in detail and identified potential risks in the project



Step 2.4) Create Test Logistics

In Test Logistics, the Test Manager should answer the following questions:

Who will test?

When will the test occur?

Who will test?

You may not know exact names of the tester who will test, but the type of tester can be defined.

To select the right member for specified task, you have to consider if his skill is qualified for the task or not, also estimate the project budget. Selecting wrong member for the task may cause the project to fail or delay.

Person having the following skills is most ideal for performing software testing:

Ability to understand customers point of view

Strong desire for quality

Attention to detail

Good cooperation

In your project, the member who will take in charge for the test execution is the tester. Base on the project budget, you can choose in-source or outsource member as the tester.

When will the test occur?

Test activities must be matched with associated development activities

Step 3) Define Test Objective

Test Objective is the overall goal and achievement of the test execution. The objective of the testing is finding as many software defects as possible; ensure that the software under test is bug free before release.

To define the test objectives, you should do 2 following steps

List all the software features (functionality, performance, GUI…) which may need to test.

Define the target or the goal of the test based on above features

Let’s apply these steps to find the test objective of your Guru99 Bank testing project

You can choose the ‘TOP-DOWN’ method to find the website’s features which may need to test. In this method, you break down the application under test to component and sub-component.

Based on above features, you can define the Test Objective of the project Guru99 as following

Check that whether website Guru99 functionality(Account, Deposit…) is working as expected without any error or bugs in real business environment

Check that the external interface of the website such as UI is working as expected and & meet the customer need

Verify the usability of the website. Are those functionalities convenient for user or not?

Step 4) Define Test Criteria

Test Criteria is a standard or rule on which a test procedure or test judgment can be based. There’re 2 types of test criteria as following

Suspension Criteria

Specify the critical suspension criteria for a test. If the suspension criteria are met during testing, the active test cycle will be suspended until the criteria are resolved.

Test Plan Example: If your team members report that there are 40% of test cases failed, you should suspend testing until the development team fixes all the failed cases.

Exit Criteria

It specifies the criteria that denote a successful completion of a test phase. The exit criteria are the targeted results of the test and are necessary before proceeding to the next phase of development. Example: 95% of all critical test cases must pass.

Some methods of defining exit criteria are by specifying a targeted run rate and pass rate.

Run rate is ratio between number test cases executed/total test cases of test specification. For example, the test specification has total 120 TCs, but the tester only executed 100 TCs, So the run rate is 100/120 = 0.83 (83%)

Pass rate is ratio between numbers test cases passed / test cases executed. For example, in above 100 TCs executed, there’re 80 TCs that passed, so the pass rate is 80/100 = 0.8 (80%)

This data can be retrieved in Test Metric documents.

Run rate is mandatory to be 100% unless a clear reason is given.

Pass rate is dependent on project scope, but achieving high pass rate is a goal.

Step 5) Resource Planning

Resource plan is a detailed summary of all types of resources required to complete project task. Resource could be human, equipment and materials needed to complete a project

The resource planning is important factor of the test planning because helps in determining the number of resources (employee, equipment…) to be used for the project. Therefore, the Test Manager can make the correct schedule & estimation for the project.

Step 6) Plan Test Environment

What is the Test Environment

A testing environment is a setup of software and hardware on which the testing team is going to execute test cases. The test environment consists of real business and user environment, as well as physical environments, such as server, front end running environment.

How to setup the Test Environment

Step 7) Schedule & Estimation

In the article [Test estimation](https://www.guru99.com/an-expert-view-on-test-estimation.html), you already used some techniques to estimate the effort to complete the project. Now you should include that estimation as well as the schedule to the Test Planning

In the Test Estimation phase, suppose you break out the whole project into small tasks and add the estimation for each task as below

Step 8) Test Deliverables

Test Deliverables is a list of all the documents, tools and other components that has to be developed and maintained in support of the testing effort.

Test deliverables are provided before testing phase.

Test plans document.

Test cases documents

Test Design specifications.

Test deliverables are provided during the testing

Test Scripts

Simulators.

Test Data

Test Traceability Matrix

Error logs and execution logs.

Test deliverables are provided after the testing cycles is over.

Test Results/reports

Defect Report

Installation/ Test procedures guidelines

Release notes

What are the different Methodologies in Agile Development Model

What are the Different Types of Agile Methodologies?

Agile refers to the methods and best practices for organizing projects based on the values and principles documented in the [Agile Manifesto](https://www.agilealliance.org/agile101/what-is-agile/). However, there’s no one right way to implement Agile and many different types of methodologies from which to choose. Here are some of the most common Agile frameworks.

Kanban

Kanban is a simple, visual means of managing projects that enables teams to see the progress so far and what’s coming up next. Kanban projects are primarily managed through a Kanban board, which segments tasks into three columns: “To Do,” “Doing,” and “Done.”

Scrum

Scrum is similar to Kanban in many ways. Scrum typically uses a Scrum board, similar to a Kanban board, and groups tasks into columns based on progress. Unlike Kanban, Scrum focuses on breaking a project down into sprints and only planning and managing one sprint at a time. Scrum also has unique project roles: Scrum master and product owner.

Extreme Programming (XP)

Extreme Programming (XP) was designed for Agile software development projects. It focuses on continuous development and customer delivery and uses intervals or sprints, similar to a Scrum methodology. However, XP also has 12 supporting processes specific to the world of software development:

Planning game

Small releases

Customer acceptance tests

Simple design

Pair programming

Test-driven development

Refactoring

Continuous integration

Collective code ownership

Coding standards

Metaphor

Sustainable pace

Feature-driven development (FDD)

Feature-driven development is another software-specific Agile framework. This methodology involves creating software models every two weeks and requires a development and design plan for every model feature. It has more rigorous documentation requirements than XP, so it’s better for teams with advanced design and planning abilities. FDD breaks projects down into five basic activities:

Develop an overall model

Build a feature list

Plan by feature

Design by feature

Build by feature

Dynamic Systems Development Method (DSDM)

The Dynamic Systems Development Method (DSDM) was born of the need for a common industry framework for rapid software delivery. Rework is to be expected, and any development changes that occur must be reversible. Like Scrum, XP, and FDD, DSDM uses sprints. This framework is based on eight fundamental principles:

Focus on the business need

Deliver on time

Collaborate

Never compromise quality

Build incrementally from firm foundations

Develop iteratively

Communicate continuously and clearly

Demonstrate control

Crystal

Crystal is a family of Agile methodologies that includes Crystal Clear, Crystal Yellow, Crystal Orange, Crystal Red, etc. Each has a unique framework. Your choice depends on several project factors, such as your team size, priorities, and project criticality.

Lean

Lean development is often grouped with Agile, but it’s an entirely different methodology that happens to share many of the same values. The [main principles of the Lean methodology](https://leankit.com/learn/lean/principles-of-lean-development/) include:

Eliminating waste

Build quality in

Create knowledge

Defer commitment

Deliver fast

Respect people

Optimize the whole

Explain the difference between Authorization and Authentication in Web testing

Authentication vs. Authorization

So, what is the difference between authentication and authorization? Simply put, authentication is the process of verifying who someone is, whereas authorization is the process of verifying what specific applications, files, and data a user has access to. The situation is like that of an airline that needs to determine which people can come on board. The first step is to confirm the identity of a passenger to make sure they are who they say they are. Once a passenger’s identity has been determined, the second step is verifying any special services the passenger has access to, whether it’s flying first-class or visiting the VIP lounge.

In the digital world, authentication and authorization accomplish these same goals. Authentication is used to verify that users really are who they represent themselves to be. Once this has been confirmed, authorization is then used to grant the user permission to access different levels of information and perform specific functions, depending on the rules established for different types of users.

|  |  |
| --- | --- |
| **Authentication** | **Authorization** |
| Authentication verifies who the user is. | Authorization determines what resources a user can access. |
| Authentication works through passwords, one-time pins, biometric information, and other information provided or entered by the user. | Authorization works through settings that are implemented and maintained by the organization. |
| Authentication is the first step of a good identity and access management process. | Authorization always takes place after authentication. |
| Authentication is visible to and partially changeable by the user. | Authorization isn’t visible to or changeable by the user. |
| Example: By verifying their identity, employees can gain access to an HR application that includes their personal pay information, vacation time, and 401K data. | Example: Once their level of access is authorized, employees and HR managers can access different levels of data based on the permissions set by the organization. |

Common Authentication Methods

While user identity has historically been validated using the combination of a username and password, today’s authentication methods commonly rely upon three classes of information:

What you know: Most commonly, this is a password. But it can also be an answer to a security question or a one-time pin that grants user access to just one session or transaction.

What you possess: This could be a mobile device or app, a security token, or digital ID card.

What you are: This is biometric data such as a fingerprint, retinal scan, or facial recognition.

Oftentimes, these types of information are combined using [multiple layers of authentication](https://www.sailpoint.com/identity-library/authentication-methods-used-for-network-security/). For example, a user may be asked to provide a username and password to complete an online purchase. Once that’s confirmed, a one-time pin may be sent to the user’s mobile phone as a second layer of security. Combining multiple authentication methods with consistent [authentication protocols](https://www.sailpoint.com/identity-library/identity-management-protocols/), organizations can ensure security as well as compatibility between systems.

Common Authorization Methods

Once a user is authenticated, authorization controls are then applied to ensure users can access the data they need and perform specific functions such as adding or deleting information—based on the permissions granted by the organization. These permissions can be assigned at the application, operating system, or infrastructure levels. Two common authorization techniques include:

[Role-based access controls (RBAC)](https://www.sailpoint.com/identity-library/what-is-role-based-access-control/): This authorization method gives users access to information based on their role within the organization. For example, all employees within a company may be able to view, but not modify, their personal information such as pay, vacation time, and 401K data. Yet HR managers may be given access to all employees’ HR information with the ability to add, delete, and change this data. By assigning permissions according to each person’s role, organizations can ensure every user is productive, while limiting access to sensitive information.

[Attribute-based access control (ABAC)](https://www.sailpoint.com/identity-library/what-is-attribute-based-access-control/): ABAC grants users permissions on a more granular level than RBAC using a series of specific attributes. This may include user attributes such as the user’s name, role, organization, ID, and security clearance. It may include environmental attributes such as the time of access, location of the data, and current organizational threat levels. And it may include resource attributes such as the resource owner, file name, and level of data sensitivity. ABAC is a more complex authorization process than RBAC designed to further limit access. For example, rather than allowing all HR managers in an organization to change employees’ HR data, access can be limited to certain geographical locations or hours of the day to maintain tight security limits.

A Strong Authentication & Authorization Strategy Is Essential

A sound security strategy requires protecting one’s resources with both authentication and authorization. With a strong authentication and authorization strategy in place, organizations can consistently verify who every user is and what they have access to do—preventing unauthorized activity that poses a serious threat. By ensuring all users properly identify themselves and access only the resources they need, organizations can maximize productivity, while bolstering their security at a time when data breaches are robbing businesses of their revenue and their reputation.

See how [SailPoint integrates with the right authentication providers](https://www.sailpoint.com/integrations/access-management/).

What are the common problems faced in Web testing

With the bar of customer expectations raised higher than ever before, make sure avoidable issues aren’t damaging your brand’s revenue or reputation.

It goes without saying that quality matters when you launch or update a website or mobile app. Five years ago, we had to make the case for digital quality by convincing businesses of its impact on revenue, spelling out the connection between customer experience and customer spending and revisits. Nowadays, most of our clients understand very clearly that customers vote with their feet when digital quality is poor, and know that it can make a massive dent in revenue and profits.

These days all of us are consumers, demanding our digital and mobile apps deliver simple, flawless experiences whether we’re using an office application, a banking platform, a social media app or an ecommerce site. That means that the bar is at an all-time high, when it comes to meeting digital quality expectations. Customers have a very short fuse and will walk away with little provocation, anticipating that there’s a competitor round the virtual corner who’s hungry for their business and won’t make their digital life inconvenient.

It’s more important than ever to eradicate avoidable issues when developing your digital assets. That’s why in this blog we’re sharing the top ten app and web testing issues that we tackle for our clients every day. There are straightforward fixes for all these common issues in digital projects: take a look through our hotlist to check that you’re not falling into any of these common traps in your [web and mobile app testing](https://www.digivante.com/services/testing-capabilities/).

1 – Insufficient testing for browser compatibility

Poor [browser compatibility](https://www.digivante.com/services/testing-capabilities/cross-browser-testing/) is an obvious turn-off – if users can’t see your content clearly and transactional features don’t work well, they’ll move on.

You need to make sure that your website supports all browsers that your visitors use to access your site. Use tools like Google Analytics to get an idea of what the most popular browsers are, so that you can prioritise compatibility testing for your website and as a bare minimum, be sure that the most popular browsers are rigorously assessed.

But it’s not just a numbers game. You can’t afford to alienate customers who may have a potentially high value. And user preferences change. So you do also need to test less popular or niche browsers, as well as older versions of popular browsers.

Not everyone will be using Chrome, Edge or Firefox. Your site needs to work consistently across as many browsers as possible so you can engage and retain visitors to your site. If a visitor to your site experiences an issue, they probably won’t continue using it, nor spend any money on a purchase they have to struggle to make. Moreover, they might tell others about their poor experience. And you’ll certainly miss out on any positive advocacy they might have offered if they’d had a great experience on their chosen browser.

2 – Failing to conduct thorough functional testing across mobile

Many websites and apps suffer from functional issues on different mobile devices, because they haven’t been tested on all the devices their customers are using.

Nowadays, it’s widely acknowledged that mobile is the most common way for many users to access websites. Testing has historically prioritised desktop devices but.

In 2022, there are 6.64 billion smartphones in use worldwide – well over 80% of the population. 90% of internet users go online with a smartphone. And smartphone usage is continually increasing – it’s estimated that there will be a billion more by 2027. Thorough functional testing across mobile is essential.

There are three issues that mobile users regularly face. First, poorly functioning mobile apps. Speed is a particular issue, with users likely to abandon a mobile app if it fails to load within three seconds. Secondly, a consistent experience. Mobile users should be able to access the same functionality that desktop users enjoy. This includes sign-up and log-in pages, menus and consistent rendering across varying screen sizes and resolutions. Thirdly, carrier variations. It’s important to test how your app performs on different networks too. They each have their own configurations: just because your website works well on one mobile carrier doesn’t mean it will on another.

3 – Failing to conduct thorough functional testing across desktop

Yes, it’s the same problem but in reverse. Some companies are so carried away with the news that mobile is predominant that they drop the ball on desktop testing. Responsive mobile design has become the default for many development teams. But although desktop usage may be in the minority, crucial segments of your target audience may over-index on desktop. Some users prefer browsing and reading on a larger screen. Many others will consume information and consider purchases across multiple devices on their user journey – inconsistency in the experience can undermine their trust in your brand. And the trends are different in different markets and industries.

If you allow more affluent or discerning users to experience glitches and poor function, it can hit revenue and reputation hard. Make sure you’ve looked at your user profiles carefully before pulling the rug out from desktop testing, or your could be making a catastrophically expensive mistake.

4 – Poor data security

Data security is a constant challenge for organisations to deal with. Hackers regularly probe company security for weaknesses. New viruses and malware are constantly being developed to exploit unpatched software vulnerabilities. Paying insufficient attention to security in your QA testing is a big risk. You could lose revenue, compromise data and destroy customer confidence if you leave cracks open for malware or criminals to invade.

Proper cybersecurity means following strict and up-to-date practices to stop attacks. Failure to do so will almost certainly result in your organisation’s security being compromised – retail cybercrime was up 33% in 2021. The potential damage to your company’s systems is one big problem with that. But data breaches can put you on the wrong side of compliance legislation – attracting hefty fines in some sectors – as well as causing massive damage to the trust between a brand and its clients.

A 2021 report by Paysafe revealed that 44% of consumers didn’t feel comfortable entering financial details to pay online. By taking systematic steps to ensure both new and existing customers are confident in your payment and financial security, you’ll encourage them to complete purchases and to return for a reassuringly similar experience next time.

5 – Failing to provide an intuitive experience

While it’s important that your website’s visual design is up-to-date with the latest trends, a HubSpot report reveals why a large number of consumers view easy navigation as the most important factor in website design.

Clear and easy navigation guides site visitors to the information they’re looking for, quickly and easily. If they can’t find what they want, they’ll inevitably go elsewhere.

Common problems that can negatively affect website navigation are:

Broken links

Slow loading speeds

Complicated menu structures

Avoiding these problems goes right back to the start of the process: you need a website design approach with a clear vision for self-explanatory user navigation.

How will you know if you’ve succeeded with your design? A website or mobile application can be considered intuitive once a user is able to navigate it without having to stop and think what the next step is. Thorough user testing before launch or when you make changes, using testers whose profiles match your target audience as closely as possible, is the best way to zoom in on any awkward aspects of your user journeys.

6 – Not performing testing frequently enough

Tick off your pre-launch web testing and you can tick off any future problems right? Of course, this is not the case. You need to repeat website testing regularly to identify any conversion blockers that are costing you revenue, as well as any potential issues that appear over time before they become critical.

Most of us know this, but unless you schedule regular repeat testing and fence the budget and resources, it’s all too easy for ongoing testing to drop off the agenda. Testing new products and features tends to take priority. It’s vital to keep a balance, or customer experiences can be eroded over time, with insignificant changes or evolving user preferences making far more difference than you realise.

Functional testing also needs to be conducted continuously: new devices and browsers are constantly released into the marketplace and your digital products need to keep up.

7 – Leaving digital accessibility to the last minute

If you don’t take digital accessibility seriously when you’re creating websites and apps, you’re missing out on a share of the £249 billion spent annually by customers with disabilities and impairments that can affect the way they engage with digital products. You could also risk breaching equality regulations for your sector or country, as well as achieving some very negative PR, if you don’t strive to meet the WCAG standards on digital accessibility. [WCAG 2.2](https://www.digivante.com/blog/wcag-explained-future-web-accessibility/) will adds nine new areas to address.

The best time to factor in [accessibility for your website or app](https://www.digivante.com/services/testing-capabilities/accessibility-testing/), is right from the start. It’s more costly to make retroactive changes than to build them into the initial design.

Find out more about WCAG standards and best practice approaches to accessibility testing in this detailed guide: [Everything you need to know about accessibility testing](https://www.digivante.com/resource/everything-accessibility-testing/)

8 – Releasing new features breaks the existing live system

After any deployment or release, there’s a risk of unintended consequences. Changes can introduce defects or alter the behaviour of a flow or specific function.

It could be bug fixes, new features or configuration changes – altering anything can potentially have impacts in other areas of your digital product, and it’s almost impossible to anticipate them all when you’re working on a sophisticated app or website.

You need to carry out [regression testing](https://www.digivante.com/services/testing-capabilities/regression-testing/) to check that your entire system continues to function as it should. Pay particular attention to those areas that were not directly changed in the latest deployment. Work with your testing team to develop regression testing packs that thoroughly review your entire system, so you can quickly identify any areas where changes have had an adverse impact.

Find out more about regression testing in this detailed article: [How to maximise the effectiveness and efficiency of your software regression testing](https://www.digivante.com/blog/maximise-effectiveness-software-regression-testing/).

9 – Localisation and the global experience

Launching digital products to a worldwide audience, rather than one geography or in a single language, brings a whole lot more complexity. Customers and users of your software or website may interact in a language different from the one it was written in. Automated translations have come a long way, but they still don’t pick up on the subtleties of language.

It’s not only mistranslations that can issues, but also different interpretations of content by different audiences, due to cultural context. Something that seems logical and intuitive to the person who created it might not make any sense to the user, or worse, could be accidentally offensive.

To avoid these pitfalls, which can at worst hit the headlines as examples of organisations being tone deaf in new markets, you need to test your apps and websites with real people who [live, work and transact in the regions](https://www.digivante.com/services/testing-capabilities/localisation-testing/) and cultures where you’re releasing them.

Find out more about [localisation testing: Global reach with local expertise](https://www.digivante.com/blog/globalisation-and-localisation-testing/).

10 – The most common bugs

Bugs in code and errors in set-up can ruin website or app user experiences. As well as getting in the way of smooth navigation and stopping users completing transactions, they give an impression of carelessness or a lack of attention to detail. Clearly, that doesn’t play well for most brands, especially if site visitors are trusting you with their personal or payment data ,or relying on you to deliver a product or service. Here are some of the most common bugs:

Crash – caused when a software solution, operating system or program no longer works properly and forces itself to shut down. A catastrophic error that plagues apps in particular.

Functional error – when the behaviour of the software deviates from the expectation of the user. For example, if their login details aren’t recognised despite being correct.

Acknowledgement error – when a user receives feedback or responses that have the wrong message, or they receive no message at all. For example, when a user signs up for a newsletter and needs to validate their email address, they don’t receive the email with the link in it to do this.

Typos – misspellings, missed words, odd characters appearing in text… if you haven’t checked to pick these up, customers wonder what other details you might have missed.

Control flow error – these prevent software from proceeding to the next tasks in the correct way and can prevent website conversions. For example, if a user clicks the “Go to Checkout” button at the end of an online shopping experience but is not redirected to the checkout to complete the purchase.

All of these issues are avoidable, with a combination of careful planning and development, alongside – crucially – rigorous professional testing. Your in-house testing team has its work cut out to keep on top of regular and pre-launch testing in fast-moving ecommerce environments. Talk to Digivante if you need a [trusted partner to supplement internal testing resources](https://www.digivante.com/services/team-augmentation/) or would like to outsource your testing completely. Our testing specialists and expert [crowdtesting community](https://www.digivante.com/crowdtesting-community/) can help you eradicate avoidable issues with thorough and precise testing and clear actionable reporting.

If you’d like to find out more about improving your testing approach, upweighting your resources with experienced testing leads, accessing a trusted army of vetted crowdtesters, or hiring a fully outsourced QA team, [get in touch](https://www.digivante.com/contact-us) to discuss how we can help you.