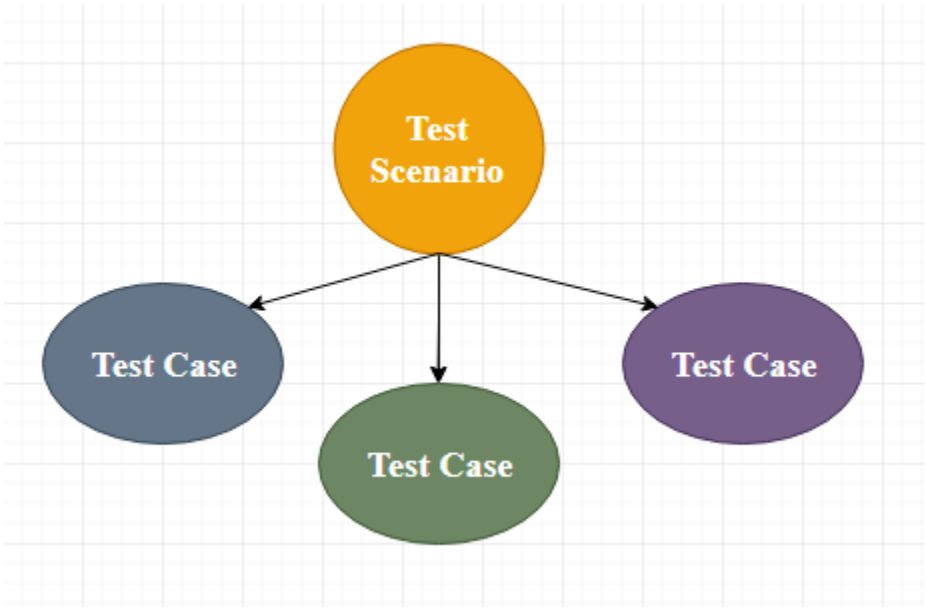
Test Case

The test case is defined as a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not. Test case designing includes preconditions, case name, input conditions, and expected result. A test case is a first level action and derived from test scenarios.



It is an in-details document that contains all possible inputs (positive as well as negative) and the navigation steps, which are used for the test execution process. Writing of test cases is a one-time attempt that can be used in the future at the time of regression testing.

Test case gives detailed information about testing strategy, testing process, preconditions, and expected output. These are executed during the testing process to check whether the software application is performing the task for that it was developed or not.

Test case helps the tester in defect reporting by linking defect with test case ID. Detailed test case documentation works as a full proof guard for the testing team because if developer missed something, then it can be caught during execution of these full-proof test cases.

To write the test case, we must have the requirements to derive the inputs, and the test scenarios must be written so that we do not miss out on any features for testing. Then we should have the test case template to maintain the uniformity, or every test engineer follows the same approach to prepare the test document.

Generally, we will write the test case whenever the developer is busy in writing the code.

When do we write a test case?

We will write the test case when we get the following:

- When the customer gives the business needs then, the developer starts developing and says that they need 3.5 months to build this product.
- And In the meantime, the testing team will **start writing the test cases**.
- Once it is done, it will send it to the Test Lead for the review process.
- And when the developers finish developing the product, it is handed over to the testing team.
- The test engineers never look at the requirement while testing the product document because testing is constant and does not depends on the mood of the person rather than the quality of the test engineer.

> *Note: When writing the test case, the actual result should never be written as the product is still being in the development process. That?s why the actual result should be written only after the execution of the test cases.*

Why we write the test cases?

We will write the test for the following reasons:

- **To require consistency in the test case execution**
- **To make sure a better test coverage**
- **It depends on the process rather than on a person**
- **To avoid training for every new test engineer on the product**

To require consistency in the test case execution: we will see the test case and start testing the application.

To make sure a better test coverage: for this, we should cover all possible scenarios and document it, so that we need not remember all the scenarios again and again.

It depends on the process rather than on a person: A test engineer has tested an application during the first release, second release, and left the company at the time of third release. As the test engineer understood a module and tested the application thoroughly by deriving many values. If the person is not there for the third release, it becomes difficult for the new person. Hence all the derived values are documented so that it can be used in the future.

To avoid giving training for every new test engineer on the product: When the test engineer leaves, he/she leaves with a lot of knowledge and scenarios. Those scenarios should be documented so that the new test engineer can test with the given scenarios and also can write the new scenarios.

Types of test cases

We have a different kind of test cases, which are as follows:

- **Function test cases**
- **Integration test cases**
- **System test cases**

The functional test cases

Firstly, we check for which field we will write test cases and then describe accordingly.

In functional testing or if the application is data-driven, we require the input column else; it is a bit time-consuming.

Rules to write functional test cases:

- In the expected results column, try to use **should be** or **must be**.
- Highlight the Object names.
- We have to describe only those steps which we required the most; otherwise, we do not need to define all the steps.
- To reduce the excess execution time, we will write steps correctly.
- Write a generic test case; do not try to hard code it.

Integration test case

In this, we should not write something which we already covered in the functional test cases, and something we have written in the integration test case should not be written in the system test case again.

Rules to write integration test cases

- Firstly, understand the product
- Identify the possible scenarios
- Write the test case based on the priority

When the test engineer writing the test cases, they may need to consider the following aspects:

If the test cases are in details:

- They will try to achieve maximum test coverage.
- All test case values or scenarios are correctly described.
- They will try to think about the execution point of view.
- The template which is used to write the test case must be unique.

System test cases

We will write the system test cases for the end-to-end business flows. And we have the entire modules ready to write the system test cases.

The process to write test cases

The method of writing a test case can be completed into the following steps, which are as below:

System study

In this, we will understand the application by looking at the requirements or the SRS, which is given by the customer.

Identify all scenarios:

- When the product is launched, what are the possible ways the end-user may use the software to identify all the possible ways.
- I have documented all possible scenarios in a document, which is called test design/high-level design.
- The test design is a record having all the possible scenarios.

Write test cases

Convert all the identified scenarios to test claims and group the scenarios related to their features, prioritize the module, and write test cases by applying test case design techniques and use the standard test case template, which means that the one which is decided for the project.

Review the test cases

Review the test case by giving it to the head of the team and, after that, fix the review feedback given by the reviewer.

Test case approval

After fixing the test case based on the feedback, send it again for the approval.

Store in the test case repository

After the approval of the particular test case, store in the familiar place that is known as the test case repository.

Reliability Testing is a testing technique that relates to test the ability of a software to function and given environmental conditions that helps in uncovering issues in the software design and functionality. It is defined as a type of software testing that determines whether the software can

perform a failure free operation for a specific period of time in a specific environment. It ensures that the product is fault free and is reliable for its intended purpose.

Objective of Reliability Testing:

The objective of reliability testing is:

- To find the perpetual structure of repeating failures.
- To find the number of failures occurring is the specific period of time.
- To discover the main cause of failure.
- To conduct performance testing of various modules of software product after fixing defects.

Types of Reliability Testing:

There are three types of reliability testing:-

1. **Feature Testing:**
   Following three steps are involved in this testing:
   - Each function in the software should be executed at least once.
   - Interaction between two or more functions should be reduced.
   - Each function should be properly executed.
2. **Regression Testing:**
   Regression testing is basically performed whenever any new functionality is added, old functionalities are removed or the bugs are fixed in an application to make sure with introduction of new functionality or with the fixing of previous bugs, no new bugs are introduced in the application.
3. **Load Testing:**
   Load testing is carried out to determine whether the application is supporting the required load without getting breakdown. It is performed to check the performance of the software under maximum work load.

Debugging

In the development process of any software, the software program is religiously tested, troubleshot, and maintained for the sake of delivering bug-free products. There is nothing that is error-free in the first go.

So, it's an obvious thing to which everyone will relate that as when the software is created, it contains a lot of errors; the reason being nobody is perfect and getting error in the code is not an issue, but avoiding it or not preventing it, is an issue!

All those errors and bugs are discarded regularly, so we can conclude that debugging is nothing but a process of eradicating or fixing the errors contained in a software program.

Debugging works stepwise, starting from identifying the errors, analyzing followed by removing the errors. Whenever a software fails to deliver the result, we need the software tester to test the application and solve it.

Since the errors are resolved at each step of debugging in the software testing, so we can conclude that it is a tiresome and complex task regardless of how efficient the result was.

Why do we need Debugging?

Debugging gets started when we start writing the code for the software program. It progressively starts continuing in the consecutive stages to deliver a software product because the code gets merged with several other programming units to form a software product.

Following are the benefits of Debugging:

- Debugging can immediately report an error condition whenever it occurs. It prevents hampering the result by detecting the bugs in the earlier stage, making software development stress-free and smooth.
- It offers relevant information related to the data structures that further helps in easier interpretation.
- Debugging assist the developer in reducing impractical and disrupting information.
- With debugging, the developer can easily avoid complex one-use testing code to save time and energy in software development.

Steps involved in Debugging

Following are the different steps that are involved in debugging:

1. **Identify the Error:** Identifying an error in a wrong may result in the wastage of time. It is very obvious that the production errors reported by users are hard to interpret, and sometimes the information we receive is misleading. Thus, it is mandatory to identify the actual error.
2. **Find the Error Location:** Once the error is correctly discovered, you will be required to thoroughly review the code repeatedly to locate the position of the error. In general, this step focuses on finding the error rather than perceiving it.
3. **Analyze the Error:** The third step comprises error analysis, a bottom-up approach that starts from the location of the error followed by analyzing the code. This step makes it easier to comprehend the errors. Mainly error analysis has two significant goals, i.e., evaluation of errors all over again to find existing bugs and postulating the uncertainty of incoming collateral damage in a fix.
4. **Prove the Analysis:** After analyzing the primary bugs, it is necessary to look for some extra errors that may show up on the application. By incorporating the test framework, the fourth step is used to write automated tests for such areas.
5. **Cover Lateral Damage:** The fifth phase is about accumulating all of the unit tests for the code that requires modification. As when you run these unit tests, they must pass.
6. **Fix & Validate:** The last stage is the fix and validation that emphasizes fixing the bugs followed by running all the test scripts to check whether they pass.

Debugging Strategies

1. **Brute Force:** Study the system for a larger duration in order to understand the system. It helps the debugger to construct different representations of systems to be debugging depending on the need. A study of the system is also done actively to find recent changes made to the software.
2. **Backtracking:** Backward analysis of the problem which involves tracing the program backward from the location of the failure message in order to identify the region of faulty code. A detailed study of the region is conducted to find the cause of defects.
3. **Forward analysis** of the program involves tracing the program forwards using breakpoints or print statements at different points in the program and studying the results. The region where the wrong outputs are obtained is the region that needs to be focused on to find the defect.
4. **Using the past experience** of the software debug the software with similar problems in nature. The success of this approach depends on the expertise of the debugger.
5. **Cause elimination:** it introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes.

Debugging Tools

The debugging tool can be understood as a computer program that is used to test and debug several other programs. Presently, there are many public domain software such as **gdb** and **dbx** in the market, which can be utilized for debugging. These software offers console-based command-line interfaces. Some of the automated debugging tools include code-based tracers, profilers, interpreters, etc.

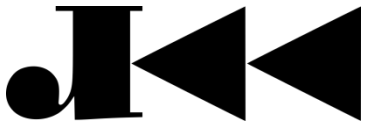
Here is a list of some of the widely used debuggers:

- Radare2
- WinDbg
- Valgrind

Radare2

Radare2 is known for its reverse engineering framework as well as binary analysis. It is made up of a small set of utilities, either utilized altogether or independently from the command line. It is also known as r2.

It is constructed around disassembler for computer software for generating assembly language source code from machine-executable code. It can support a wide range of executable formats for distinct architectures of processors and operating systems.



WinDbg

WinDbg is a multipurpose debugging tool designed for Microsoft Windows operating system. This tool can be used to debug the memory dumps created just after the Blue Screen of Death that further arises when a bug check is issued. Besides, it is also helpful in debugging the user-mode crash dumps, which is why it is called post-mortem debugging.

Valgrind

The Valgrind exist as a tool suite that offers several debugging and profiling tools to facilitate users in making faster and accurate program. Memcheck is one of its most popular tools, which can successfully detect memory-related errors caused in C and C++ programs as it may crash the program and result in unpredictable behavior.