Software Testing Fundamentals (STF) !

SOFTWARE TESTING Fundamentals (STF) is a platform to gain (or refresh) basic knowledge in the field of Software Testing. If we are to 'cliche' it, the site is **of** the testers, **by** the testers, and **for** the testers. Our goal is to build a resourceful repository of *Quality Content* on *Quality*. **YES**, you found it: the not-so-ultimate-but-fairly-comprehensive site for software testing enthusiasts.

Software Testing Definition

- **ISTQB: testing:** The process consisting of all lifecycle activities, both static and dynamic, concerned with planning, preparation and evaluation of a component or system and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.
- **STF: testing:** The practice of investigating a software / system under test so as to ensure that it is of the highest quality.

Why Software Testing?

Software that does not work correctly can lead to many problems such as:

- Delay / Loss of time
- Futility / Loss of effort
- Wastage / Loss of money
- Shame / Loss of business reputation
- Injury or death

Testing helps in ensuring that software work correctly and reduces the risk of software failure, thereby avoiding the problems mentioned above.

Software Testing Goals

The three main goals of Software Testing are:

- **Defect Detection:** Find defects / bugs in the software during all stages of its development (earlier, the better).
- **Defect Prevention:** As a consequence of defect detection, help anticipate and prevent defects from occurring at later stages of development or from recurring in the future.
- **User Satisfaction:** Ensure customers / users are satisfied that their requirements (explicit or implicit) are met.

Testing and Debugging

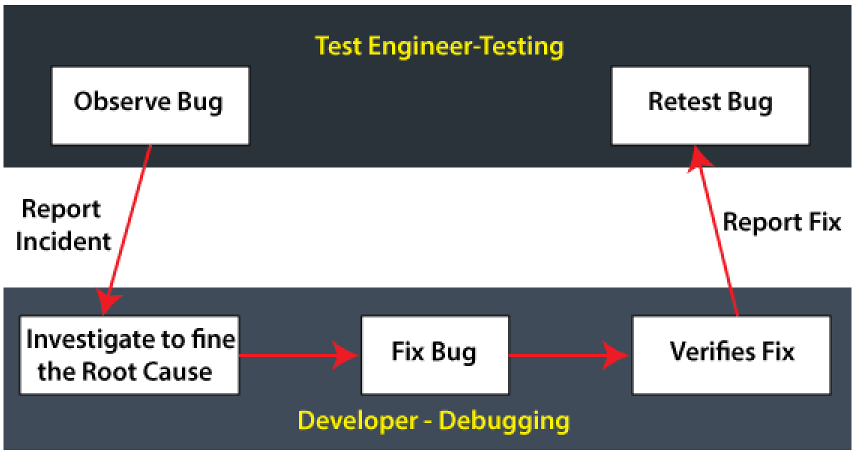
At the time of development and after the outcome of any application or the software product established in any programming language, both **Testing and Debugging** play a vital role in finding and removing mistakes.

Note: Both Testing and Debugging are two words that seem to share a similar meaning but intensively different from one another.

They have quite an equivalent function, but they are diverse in terms of **designs, requirements, benefits, and performance.**

Therefore, it is required for us to understand the **differences between testing and debugging** properly that will support us in receiving better software development outcomes.

Before we see the difference between **testing and debugging**, we will discuss the **in-detail** evaluation of **testing and debugging**, which will help us distinguish both of them appropriately.



What is Software Testing?

Software testing

is a process of identifying defects in the software product. It is performed to validate the behavior of the software or the application compared to requirements.

In other words, we can say that the testing is a collection of techniques to determine the accuracy of the application under the predefined specification but, it cannot identify all the defects of the software.

Each software or application needs to be tested before delivering to the clients and checks whether the particular software or the application is working fine as per the given requirements.

What is Debugging?

As opposed to Testing, Debugging is the action where the development team or a developer implements after receiving the test report related to the bugs in the software from the testing team.

In the software development process, debugging includes detecting and modifying code errors in a software program.

In debugging process, the developer needs to identify the reason behind the particular **bug**

or defect, which is carried out by analyzing the coding rigorously.

The developer changes the code and then rechecks whether the defect has been deleted whenever the bug or error is found.

Once the debugging is successfully finished, the application is again sent back to the test engineers, who remain in the process of testing. The debugging process allows us an earlier finding of an error and makes software development stress-free and easy.

S.NO	Testing	Debugging
1.	It is the implementation of the software with the intent of identifying the defects	The process of fixing and resolving the defects is known as debugging.
2.	Testing can be performed either manually or with the help of some automation tools.	The debugging process cannot be automated.
3.	A group of test engineers executes testing, and sometimes it can be performed by the developers.	Debugging is done by the developer or the programmer.
4.	The test engineers perform manual and automated test cases on the application, and if they detect any bug or error, they can report back to the development team for fixing.	The developers will find, evaluates, and removes the software errors.
5.	Programming knowledge is not required to perform the testing process.	Without having an understanding of the programming language, we cannot proceed with the debugging process.
6.	Once the coding phase is done, we proceed with the testing process.	After the implementation of the test case, we can start the Debugging process.
7.	Software Testing includes two or more activities such as validation and verification of the software.	Debugging tries to match indication with cause, hence leading to the error correction.
8.	It is built on different testing levels such as Unit Testing, Integration Testing, System Testing, etc.	It is built on different kinds of bugs because there is no such level of debugging is possible.
9.	Software testing is the presentation of defects.	It is a logical procedure.
10.	Software testing is the vital phase of SDLC (Software Development Life Cycle).	It is not a part of SDLC because it occurs as a subset of testing.
11.	Some advantages of software testing are as below: ○ It can easily understand by the new test engineers or the beginner. ○ The test engineer can interact with software as a **real end-user** to check the usability and user interface issues. ○ It is used to test dynamically altering GUI designs. ○ Testing is a cost-effective and time-saving process. ○ Software testing delivers a **consistence software.** ○ It will help us to execute the root cause analysis that will enhance the software's productivity. ○ The testing process also helps detect and fixing the bugs before the software becomes active, which significantly reduces the risk of failure.	Some advantages of debugging process are as follows: ○ It supports the developer in minimizing the data. ○ If the perform the debugging, we can report the error condition directly. ○ During the debugging process, the developer can **avoid complex one-use testing code** thathelps the developer save time and energy. ○ Debugging delivers maximum useful information of data structures and allows its informal understanding.

12.	Software testing contains various type of testing methods, which are as follow: ○ Black-box testing ○ White-box testing ○ Grey-box testing And some other type of testing types is as below: ○ Unit testing ○ Integration Testing ○ System Testing ○ Stress Testing ○ Performance Testing ○ Compatibility Testing ○ Beta Testing ○ Alpha Testing ○ Smoke Testing ○ Regression Testing ○ User Acceptance Testing and so on.	Debugging involves a various type of approaches, which are as follows: ○ Induction ○ Brute Force ○ Deduction
13.	The testing team can subcontract to the outside team as well.	Debugging cannot be subcontracted to an outside team because the inside development team only does it.
14.	We can plan, design, and implement the testing process.	As compared to the testing process, the debugging process cannot be forced.

Software Quality

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc.for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

The modern view of a quality associated with a software product several quality methods such as the following:

Portability: A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.

Usability: A software product has better usability if various categories of users can easily invoke the functions of the product.

Reusability: A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

Correctness: A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

Maintainability: A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

Software Quality Management System

A quality management system is the principal methods used by organizations to provide that the products they develop have the desired quality.

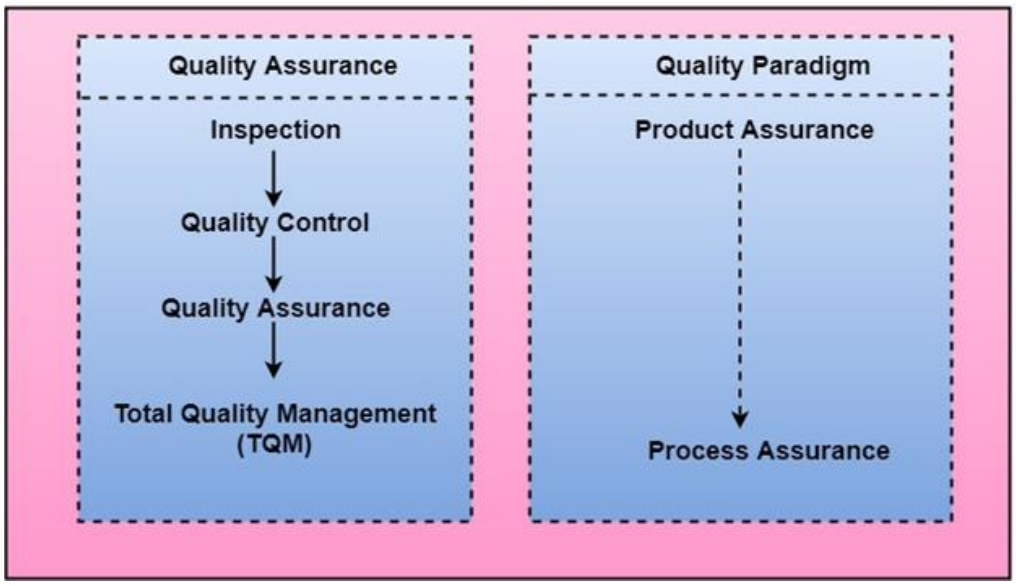
A quality system subsists of the following:

Managerial Structure and Individual Responsibilities: A quality system is the responsibility of the organization as a whole. However, every organization has a sever quality department to perform various quality system activities. The quality system of an arrangement should have the support of the top management. Without help for the quality system at a high level in a company, some members of staff will take the quality system seriously.

Quality System Activities: The quality system activities encompass the following:

- Auditing of projects
- Review of the quality system
- Development of standards, methods, and guidelines, etc.
- Production of documents for the top management summarizing the effectiveness of the quality system in the organization.

Evolution of quality system and corresponding shift in the quality paradigm



Behaviour Testing

Behavioural Testing is a testing of the external behaviour of the program, also known as black box testing. It is usually a functional testing.

Techniques used in Black box testing

- Equivalence Class
- Boundary Value Analysis
- Domain Tests
- Orthogonal Arrays
- Decision Tables
- State Models
- Exploratory Testing
- All-pairs testing

Correctness

Correctness from software engineering perspective can be defined as the adherence to the specifications that determine how users can interact with the software and how the software should behave when it is used correctly.

If the software behaves incorrectly, it might take considerable amount of time to achieve the task or sometimes it is impossible to achieve it.

Important rules:

Below are some of the important rules for effective programming which are consequences of the program correctness theory.

- Defining the problem completely.
- Develop the algorithm and then the program logic.
- Reuse the proved models as much as possible.
- Prove the correctness of algorithms during the design phase.
- Developers should pay attention to the clarity and simplicity of your program.
- Verifying each part of a program as soon as it is developed.

Fundamental Test Process In Software Testing

Testing is a process rather than a single activity. This process starts from test planning then designing **test cases**, preparing for execution and evaluating status till the test closure. So, we can divide the activities within the fundamental test process into the following basic steps:

1) Planning and Control
2) Analysis and Design
3) Implementation and Execution
4) Evaluating exit criteria and Reporting
5) Test Closure activities

1) Planning and Control:

Test planning has following major tasks:
i. To determine the scope and **risks** and identify the objectives of testing.

ii. To determine the test approach.
iii. To implement the test policy and/or the **test strategy**. (Test strategy is an outline that describes the testing portion of the **software development cycle**. It is created to inform PM, testers and developers about some key issues of the testing process. This includes the testing objectives, method of testing, total time and resources required for the project and the testing environments.).
iv. To determine the required test resources like people, test environments, PCs, etc.
v. To schedule test analysis and design tasks, test implementation, execution and evaluation.
vi. To determine the **Exit criteria** we need to set criteria such as **Coverage criteria.** (Coverage criteria are the percentage of statements in the software that must be executed during testing. This will help us track whether we are completing test activities correctly. They will show us which tasks and checks we must complete for a particular level of testing before we can say that testing is finished.)

Test control has the following major tasks:
i. To measure and analyze the results of reviews and testing.
ii. To monitor and document progress, **test coverage** and exit criteria.
iii. To provide information on testing.
iv. To initiate corrective actions.
v. To make decisions.

2) Analysis and Design:

Test analysis and Test Design has the following major tasks:
i. To review the **test basis.** (The test basis is the information we need in order to start the test analysis and create our own test cases. Basically it's a documentation on which test cases are based, such as requirements, design specifications, product risk analysis, architecture and interfaces. We can use the test basis documents to understand what the system should do once built.)
ii. To identify test conditions.
iii. To design the tests.
iv. To evaluate testability of the requirements and system.
v. To design the test environment set-up and identify and required infrastructure and tools.

3) Implementation and Execution:

During test implementation and execution, we take the test conditions into **test cases** and procedures and other **testware** such as scripts for automation, the test environment and any other test infrastructure. (Test cases is a set of conditions under which a tester will determine whether an application is working correctly or not.)
(Testware is a term for all utilities that serve in combination for testing a software like scripts, the test environment and any other test infrastructure for later reuse.)

Test implementation has the following major task:
i. To develop and prioritize our test cases by using techniques and create **test data** for those tests. (In order to test a software application you need to enter some data for testing most of the features. Any such specifically identified data which is used in tests is known as test data.)
We also write some instructions for carrying out the tests which is known as **test procedures.**
We may also need to automate some tests using **test harness** and automated tests scripts. (A test harness is a collection of software and test data for testing a program unit by running it under different conditions and monitoring its behavior and outputs.)
ii. To create test suites from the test cases for efficient test execution.
(Test suite is a collection of test cases that are used to test a software program to show that it has some specified set of behaviours. A test suite often contains detailed instructions and information for each collection of test cases on the system configuration to be used during testing. Test suites are used to group similar test cases together.)
iii. To implement and verify the environment.

Test execution has the following major task:
i. To execute test suites and individual test cases following the test procedures.
ii. To re-execute the tests that previously failed in order to confirm a fix. This is known as **confirmation testing or re-testing.**
iii. To log the outcome of the test execution and record the identities and versions of the software under tests. The **test log** is used for the audit trial. (A test log is nothing but, what are the test cases that we executed, in what order we executed, who executed that test cases and what is the status of the test case (pass/fail). These descriptions are documented and called as test log.).
iv. To Compare actual results with expected results.
v. Where there are differences between actual and expected results, it report discrepancies as Incidents.

4) Evaluating Exit criteria and Reporting:

Based on the risk assessment of the project we will set the criteria for each test level against which we will measure the "enough testing". These criteria vary from project to project and are known as **exit criteria**.
Exit criteria come into picture, when:
— Maximum test cases are executed with certain pass percentage.
— Bug rate falls below certain level.
— When achieved the deadlines.

Evaluating exit criteria has the following major tasks:
i. To check the test logs against the exit criteria specified in test planning.
ii. To assess if more test are needed or if the exit criteria specified should be changed.
iii. To write a test summary report for stakeholders.

5) Test Closure activities:

Test closure activities are done when software is delivered. The testing can be closed for the other reasons also like:

- When all the information has been gathered which are needed for the testing.
- When a project is cancelled.
- When some target is achieved.
- When a maintenance release or update is done.

Test closure activities have the following major tasks:
i. To check which planned deliverables are actually delivered and to ensure that all incident reports have been resolved.
ii. To finalize and archive testware such as scripts, test environments, etc. for later reuse.
iii. To handover the testware to the maintenance organization. They will give support to the software.
iv To evaluate how the testing went and learn lessons for future releases and projects.

The Psychology of Testing

1) **Error human Est :-** People make mistakes, but they do not like to admit them! One goal of testing software is to find discrepancies between the software and the specifications, or customer needs. The failures found must be reported to the developers. This section describes how the psychological problems occurring in connection with this can be dealt with. The tasks of developing software are often seen as constructive actions. The tasks of examining documents and software are seen as destructive actions. The attitudes of those involved relating to their job often differ due to this Developer test Blindness to one's own mistakes Independent test team Failure reporting perception. But these differences are not justifiable, because "testing is an extremely creative and intellectually challenging task"
2) **Developer test :-** "Can the developer test his own program?" is an important and frequently asked question. There is no universally valid answer. If the tester is also the author of the program,

she must examine her own work very critically. Only very few people are able to keep the necessary distance to a self-created product. Who really likes to detect and show their own mistakes? Developers would rather not find any defects in their own program text. The main weakness of developer tests is that developers who have to test their own programs will tend to be too optimistic. There is the danger of forgetting reasonable test cases or, because they are more interested in programming than in testing, only testing superficially.

3) **Blindness to one's own mistakes :-** If a developer implemented a fundamental design error—for example, if she misunderstood the task—then she will not find this using her own tests. The proper test case will not even come to mind. One possibility to decrease this problem of "blindness to one's own errors" is to work together in pairs and let a colleague test the programs. On the other hand, it is advantageous to have a deep knowledge of one's own test object. Time is saved because it is not necessary to learn the test object. Management has to decide when saving time is an advantage over blindness to one's own errors. This must be decided depending on the criticality of the test object and the associated failure risk.

4) **Independent test team:-** An independent testing team is beneficial for test quality and comprehensiveness. Further information on the formation of independent test teams can be found in section 6.1.1. The tester can look at the test object without bias. It is not the tester's own product, and the tester does not necessarily share possible developer assumptions and misunderstandings. The tester must, however, acquire the necessary knowledge about the test object in order to create test cases, which takes time. But the tester typically has more testing knowledge. A developer does not have this knowledge and must acquire it (or rather should have acquired it before, because the necessary time is often not unavailable during the project).

5) **Failure reporting:-** The tester must report the failures and discrepancies observed to the author and/or to management. The way this reporting is done can Mutual comprehension contribute to cooperation between developers and testers. If it's not done well, it may negatively influence the important communication of these two groups. To prove other people's mistakes is not an easy job and requires diplomacy and tact. Often, failures found during testing are not reproducible in the development environment for the developers. Thus, in addition to a detailed description of failures, the test environment must be documented in detail so that differences in the environments can be detected, which can be the cause for the different behavior. It must be defined in advance what constitutes a failure or discrepancy. If it is not clearly visible from the requirements or specifications, the customer, or management, is asked to make a decision. A discussion between the involved staff, developer, and tester as to whether this is a fault or not is not helpful. The often heard reaction of developers against any critique is, "It's not a bug, it's a feature!" That's not helpful either

6) **Mutual comprehension:-** Mutual knowledge of their respective tasks improves cooperation between tester and developer. Developers should know the basics of testing and testers should have a basic knowledge of software development. This eases the understanding of the mutual tasks and problems. The conflicts between developer and tester exist in a similar way at the management level. The test manager must report the →test results to the project manager and is thus often the messenger bringing bad news. The project manager then must decide whether there still is a chance to meet the deadline and possibly deliver software with known problems or if delivery should be delayed and additional time used for corrections. This decision depends on the severity of the failures and the possibility to work around the faults in the software.

General Principles of Testing

During the last 40 years, several principles for testing have become accepted as general rules for test work

Principle 1: Testing shows the presence of defects, not their absence.

Testing can show that the product fails, i.e., that there are defects. Testing cannot prove that a program is defect free. Adequate testing reduces the probability that hidden defects are present in the test object. Even if no failures are found during testing, this is no proof that there are no defects.

Principle 2:

Exhaustive testing is impossible.

It’s impossible to run an exhaustive test that includes all possible values for all inputs and their combinations combined with all different preconditions. Software, in normal practice, would require an “astronomically” high number of test cases. Every test is just a sample. The test effort must therefore be controlled, taking into account risk and priorities.

Principle 3:

Testing activities should start as early as possible.

Testing activities should start as early as possible in the software life cycle and focus on defined goals. This contributes to finding defects early.

Principle 4:

Defect clustering. Defects are not evenly distributed;

they cluster together. Most defects are found in a few parts of the test object. Thus if many defects are detected in one place, there are normally more defects nearby. During testing, one must react flexibly to this principle.

Principle 5:

The pesticide paradox. Insects and bacteria become resistant to pesticides.

Similarly, if the same tests are repeated over and over, they tend to loose their effectiveness: they don’t Dealing with critical information discover new defects. Old or new defects might be in program parts not executed by the test cases. To maintain the effectiveness of tests and to fight this “pesticide paradox,” new and modified test cases should be developed and added to the test. Parts of the software not yet tested, or previously unused input combinations will then become involved and more defects may be found.

Principle 6:

Testing is context dependent.

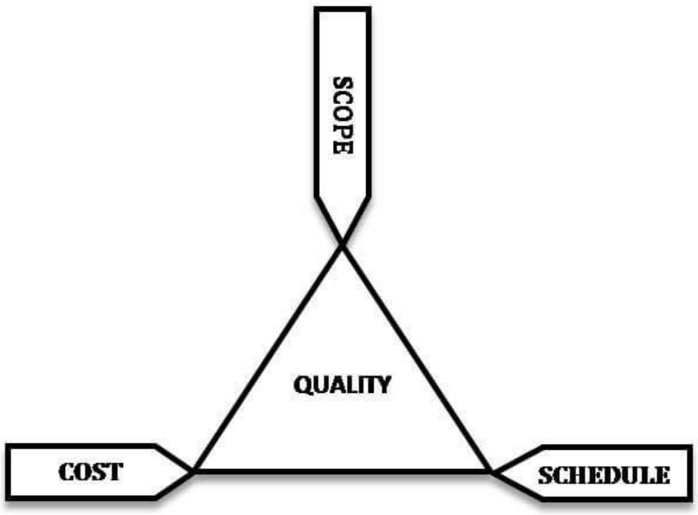
Testing must be adapted to the risks inherent in the use and environment of the application. Therefore, no two systems should be tested in the exactly same way. The intensity of testing, test exit criteria, etc. should be decided upon individually for every software system, depending on its usage environment. For example, safety-critical systems require different tests than e-commerce applications.

Principle 7: **No failures means the system is useful is a fallacy**.

Finding failures and repairing defects does not guarantee that the system meets user expectations and needs. Early involvement of the users in the development process and the use of prototypes are preventive measures intended to avoid this problem.

Software Testing Metrics

Software Testing Metrics are the quantitative measures used to estimate the progress, quality, productivity and health of the software testing process. The goal of software testing metrics is to improve the efficiency and effectiveness in the software testing process and to help make better decisions for further testing process by providing reliable data about the testing process.
A Metric defines in quantitative terms the degree to which a system, system component, or process possesses a given attribute. The ideal example to understand metrics would be a weekly mileage of a car compared to its ideal mileage recommended by the manufacturer.



Software testing metrics – Improves the efficiency and effectiveness of a software testing process.

Software testing metrics or software test measurement is the quantitative indication of extent, capacity, dimension, amount or size of some attribute of a process or product.

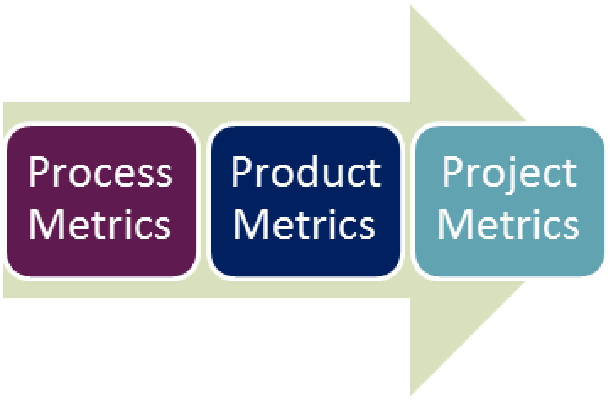
Example for software test measurement: Total number of defects

Why Test Metrics are Important?

"We cannot improve what we cannot measure" and Test Metrics helps us to do exactly the same.

- Take decision for next phase of activities
- Evidence of the claim or prediction
- Understand the type of improvement required
- Take decision or process or technology change

Types of Test Metrics



- **Process Metrics:** It can be used to improve the process efficiency of the SDLC (Software Development Life Cycle)
- **Product Metrics:** It deals with the quality of the software product
- **Project Metrics:** It can be used to measure the efficiency of a project team or any testing tools being used by the team members

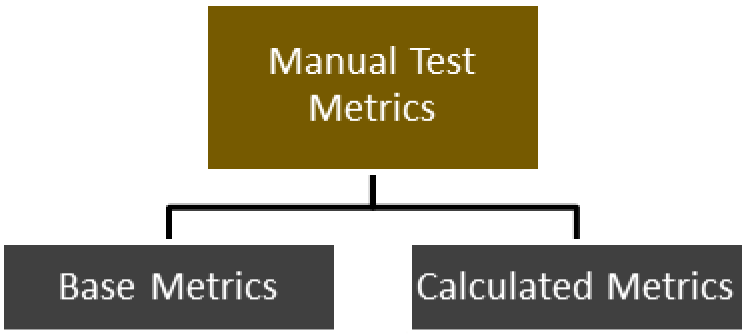
Identification of correct testing metrics is very important. Few things need to be considered before identifying the test metrics

- Fix the target audience for the metric preparation
- Define the goal for metrics
- Introduce all the relevant metrics based on project needs
- Analyze the cost benefits aspect of each metrics and the project lifestyle phase in which it results in the maximum output

Manual Test Metrics

In Software Engineering, Manual test metrics are classified into two classes

- **Base Metrics**
- **Calculated Metrics**



Base metrics is the raw data collected by Test Analyst during the test case development and execution (**# of test cases executed, # of test cases**). While calculated metrics are derived from the data collected in base metrics. Calculated metrics is usually followed by the test manager for test reporting purpose (**% Complete, % Test Coverage**).

Depending on the project or business model some of the important metrics are

- Test case execution productivity metrics
- Test case preparation productivity metrics
- Defect metrics
- Defects by priority
- Defects by severity
- Defect slippage ratio

Test Metrics Life Cycle

Different stages of Metrics life cycle	**Steps during each stage**
• Analysis	• Identification of the Metrics • Define the identified QA Metrics
• Communicate	• Explain the need for metric to stakeholder and testing team • Educate the testing team about the data points to need to be captured for processing the metric
• Evaluation	• Capture and verify the data • Calculating the metrics value using the data captured
• Report	• Develop the report with an effective conclusion • Distribute the report to the stakeholder and respective representative • Take feedback from stakeholder

How to calculate Test Metric

Sr#	**Steps to test metrics**	**Example**
1	Identify the key software testing processes to be measured	• Testing progress tracking process
2	In this Step, the tester uses the data as a baseline to define the metrics	• The number of test cases planned to be executed per day
3	Determination of the information to be followed, a frequency of tracking and the person responsible	• The actual test execution per day will be captured by the test manager at the end of the day
4	Effective calculation, management, and interpretation of the defined metrics	• The actual test cases executed per day
5	Identify the areas of improvement depending on the interpretation of defined metrics	• The Test Case execution falls below the goal set, we need to investigate the reason and suggest the improvement measures