**Five reasons why manual testing is still important**

**1. User experience**

When manual testers interact with the system as how a user would, they can identify usability issues and glitches. Human eyes can spot contextual, functional, visual, and usability bugs that automation scripts may not identify. Testing functionalities like gestures, tap, shake, captcha, and video control based trigger actions using automated scripts still need a lot of effort and time. The growing complexities of testing enhanced user experience functionalities is one of the primary reasons why manual testing is still important in agile projects.

**2. Exploratory testing**

Automation plays by the boundaries as created by test cases and scripts. Testing use cases evolve drastically as the product matures and the automation needs to keep up the pace in building test suites. Exploratory testing, a skilled form of manual testing with critical, freestyle thinking, and observational skills can discover unidentified bugs beyond coverage.

**3. Manual test reports are descriptive**

Manual testing helps us understand the problem in detail both conceptually and contextually. It allows the tester to empathize end-users while analyzing the product. Manual testers write highly descriptive test reports that explain the problem, the root cause, and suggest optimal ways to solve whereas automation test reports are simply objective.

**4. Initial investment of time and money**

Manual testing delivers faster and better quality test results when it comes to testing the early stages of a minimum viable product, and frequent updates of a matured application. On the other hand, automation testing initially takes a significant amount of time and effort in writing automated test cases and building a test automation framework.

**5. Manual testing is pocket-friendly**

For smaller projects, the ROI delivered by manual testing proves to be too good whereas automation attracts a lot of costs associated with implementing automation tools, hiring technical expertise, time and man hours spent in installation, maintenance, and subsequent management efforts.

**Software testing is broadly classified into two types:**

1. Functional testing
2. Non-functional testing

**1. Functional testing**

Functional testing is a kind of black box testing that is performed to confirm that the functionality of an application or system is behaving as expected.

 It is done to verify all the functionality of an application.

**Types in functional testing**

* Manual testing
* Automation testing
* Web service testing

**Manual testing**

Manually executes test cases (functionality) without using any automation tools is called manual testing

Manual testing is a process of finding out the defects or bugs in a software program. In this method the tester plays an important role of end user and verifies that all the features of the application are working correctly. 100 % automation is not possible.

**Automation testing**

Automated testing or test automation is a method in software testing that makes use of special software tools to control the execution of tests and then compares actual test results with predicted or expected results. ... Automation is used to add additional testing that may be too difficult to perform manually.

**Web service testing**

Sending a request and getting the response for the particular request is called web services.

Web services are those services wherein two applications or software or machines communicate with each other and exchange information, regardless of the underlying structure of the two things that are communicating.

**Non-functional testing**

Non-functional testing is done to verify the non-functional requirement of the application like performance, usability, etc.

It verifies if the behavior of the system is as per the requirement or not. It covers all the aspects which are not covered in [functional testing](https://www.softwaretestinghelp.com/guide-to-functional-testing/).

**Performance testing:**

[Evaluates the overall performance of the system](https://www.softwaretestinghelp.com/introduction-to-performance-testing-loadrunner-training-tutorial-part-1/).

* Validates that the system meets the expected response time.
* Evaluates that the significant elements of the application meet the desired response time.
* It can also be conducted as a part of integration testing and system testing.

**Load testing:**

Evaluates whether the system’s performance is as expected under normal and expected conditions.

* Validates that the system performs as expected when concurrent users access the application and get the expected response time.
* This test is repeated with multiple users to get the response time and throughput.
* At the time of testing, the database should be realistic.
* The test should be conducted on a dedicated server which stimulates the actual environment.

**Stress testing**

Evaluates whether the system’s performance is as expected when it is low on resources.

* Test on low memory or low disc space on clients/servers that reveal the defects which cannot be found under normal conditions.
* Multiple users perform the same transactions on the same data.
* Multiple clients are connected to the servers with different workloads.
* Reduce the think time to “zero” to stress the servers to their maximum stress.
* Think time: just like the time interval between typing your user and password.

**Usability testing:**

Evaluates the system for human use or checks if it is fit for use.

* Is the output correct and meaningful and is it the same as which was expected as per the business?
* Are the errors diagnosed correctly?
* Is the gui correct and consistent with the standard?
* Is the application easy for use?

### **Accessibility testing:**

Accessibility testing is defined as a type of software testing performed to ensure that the application being tested is usable by people with disabilities like hearing, color blindness, old age and other disadvantaged groups. It is a subset of [usability testing](https://www.guru99.com/usability-testing-tutorial.html).

**Seven principles of software testing**

## **1. Testing shows the presence of bugs**

Testing an application can only reveal that one or more defects exist in the application, however, testing alone cannot prove that the application is error free. Therefore, it is important to design test cases which find as many defects as possible.

## **2. Exhaustive testing is impossible**

Unless the application under test (aut) has a very simple logical structure and limited input, it is not possible to test all possible combinations of data and scenarios. For this reason, risk and priorities are used to concentrate on the most important aspects to test.

## **3. Early testing**

The sooner we start the testing activities the better we can utilize the available time. As soon as the initial products, such the requirement or design documents are available, we can start testing. It is common for the testing phase to get squeezed at the end of the development lifecycle, i.e. When development has finished, so by starting testing early, we can prepare testing for each level of the development lifecycle.

Another important point about early testing is that when defects are found earlier in the lifecycle, they are much easier and cheaper to fix. It is much cheaper to change an incorrect requirement than having to change a functionality in a large system that is not working as requested or as designed!

## **4. Defect clustering**

During testing, it can be observed that most of the reported defects are related to small number of modules within a system. I.e. Small number of modules contain most of the defects in the system. This is the application of the pareto principle to software testing: approximately 80% of the problems are found in 20% of the modules.

## **5. The pesticide paradox**

If you keep running the same set of tests over and over again, chances are no more new defects will be discovered by those test cases. Because as the system evolves, many of the previously reported defects will have been fixed and the old test cases do not apply anymore. Anytime a fault is fixed or a new functionality added, we need to do regression testing to make sure the new changed software has not broken any other part of the software. However, those regression test cases also need to change to reflect the changes made in the software to be applicable and hopefully fine new defects.

## **6. Testing is context dependent**

Different methodologies, techniques and types of testing is related to the type and nature of the application. For example, a software application in a medical device needs more testing than a games software. More importantly a medical device software requires risk based testing, be compliant with medical industry regulators and possibly specific test design techniques. By the same token, a very popular website, needs to go through rigorous performance testing as well as functionality testing to make sure the performance is not affected by the load on the servers.

## **7. Absence of errors fallacy**

Just because testing didn’t find any defects in the software, it doesn’t mean that the software is ready to be shipped. Were the executed tests really designed to catch the most defects? Or where they designed to see if the software matched the user’s requirements? There are many other factors to be considered before making a decision to ship the software.