**What is Automation Testing?**

Automation testing is a Software testing technique to test and compare the actual outcome with the expected outcome. This can be achieved by writing test scripts or using any automation testing tool. Test automation is used to automate repetitive tasks and other testing tasks which are difficult to perform manually.

Why Automation Testing?

Regression issues are the most painful issues. We are humans. And we cannot do the same thing with the same energy, speed and accuracy every day. This is what machines do. This is what automation is required for, in order to repeat the same steps with the same speed, accuracy and energy as they were repeated the first time

Whenever such a situation arises, you should automate your test case. It will help you to focus on new functionality while taking care of the regressions. With automation, you can fill that form in less than 3 minutes.

The script will fill all the fields and tell you the result along with screenshots. In case of failure, it can pinpoint the location where the test case failed, thus helping you to reproduce it with ease.

**Automation – A Cost-effective Method for Regression Testing**

Automation costs are really higher initially. It includes the cost of the tool, then the cost of the automation testing resource and their training.

But when the scripts are ready, they can be executed hundreds of times repeatedly with the same accuracy and rather quickly. This will save many hours of manual testing. So the cost gradually decreases, and ultimately it becomes a cost-effective method for [Regression testing](https://www.softwaretestinghelp.com/regression-testing-tools-and-methods/)

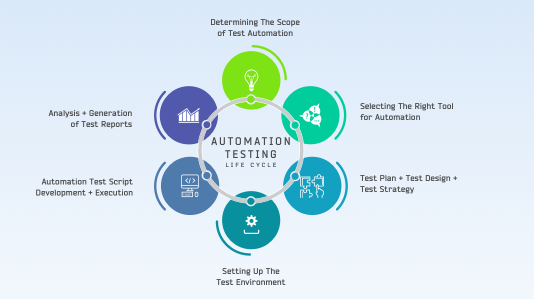
### Scenarios which require Automation

There are several situations, which cannot be tested manually and we need to automate those.

**For Example**,

1. Comparing two images pixel by pixel.
2. Comparing two spreadsheets containing thousands of rows and columns.
3. Testing an application under the load of 100,000 users.
4. Performance Benchmarks.
5. [Testing the application on different browsers](https://www.softwaretestinghelp.com/best-cross-browser-testing-tools-to-ease-your-browser-compatibility-testing-efforts/) and on different operating systems in parallel.

These situations require and should be, tested by tools.



**So, when to automate?**

This is an era of [agile methodology](https://www.softwaretestinghelp.com/agile-scrum-methodology-for-development-and-testing/) in SDLC, where the development and testing will go almost in parallel and it is very difficult to decide when to automate.

**Consider the following situations before stepping into automation**

* The product may be in its primitive stages, when the product does not even have a UI, at these stages we must have a clear thought on what we want to automate. The following points should be remembered.
  + Tests should not be obsolete.
  + As the product evolves it should be easy to pick on the scripts and add on to it.
  + It is very important to not get carried away and ensure that the scripts are easy to debug.
* Do not attempt UI automation at the very initial stages as UI is subjected to frequent changes, thereby will lead to scripts failing. As far as possible opt for API level/Non UI level automation until the product stabilizes. API automation is easy to fix and debug.

### Right Tests for Automation

The best way to tackle this problem is to quickly come up with an “Automation Strategy” that suits our product.

The idea is to group the test cases so that each group will give us a different kind of result. The Illustration given below shows how we could group our similar test cases, depending on the product/solution that we are testing.

Graphical user interface

Description automatically generated

**#1)** Make a test suite of all the basic functionality ***Positive tests***. This suite should be automated, and when this suite is run against any build, results are shown immediately. Any script failing in this suite leads to S1 or S2 defect, and that build specific can be disqualified. So we have saved a lot of time here.

As an additional step, we can add this automated test suite as a part of BVT (Build verification tests) and check the QA automation scripts into the product building process. So when the build is ready testers can check for the automation test results, and decide if the build is suitable or not for installation and further testing process.

**This clearly achieves the goals of automation which are:**

* Reduce testing effort.
* Find Bugs at earlier stages.

**#2)** Next, we have a group of ***End to End tests***.

Under large solutions, testing an end to end functionality holds the key, especially during the critical stages of the project. We should have a few automation scripts that touch upon the end to end solution tests as well. When this suite is run, the result should indicate whether the product as a whole is working as it is expected or not.

The Automation test suite should be indicated if any of the integration pieces are broken. This suite need not cover each and every small feature/functionality of the solution but it should cover the working of the product as a whole. Whenever we have an alpha or a beta or any other intermediate releases, then such scripts come in handy and give some level of confidence to the customer.

To understand better let’s assume that we are testing an ***online shopping portal***, as part of the end to end tests we should be covering only the key steps involved.

**As Given Below:**

* User login.
* Browse and select items.
* Payment Option – this covers the front end tests.
* Backend order management (involves communicating with multiple integrated partners, checking stock, emailing the user etc) – this will help the testing integration of individual pieces and also the crux of product.

So when one such script is run it gives a confidence that the solution as a whole is working fine.!

**#3)** The third set is the ***Feature/Functionality based tests***.

***For example***, We may have the functionality to browse and select a file, so when we automate this we can automate cases to include the selection of different types of files, sizes of files etc, so that feature testing is done. When there are any changes/additions to that functionality this suite can serve as a Regression suite.

**#4)** Next on the list would be **UI based tests.** We can have another suite that will test purely UI based functionalities like pagination, text box character limitation, calendar button, drop downs, graphs, images and many such UI only centric features. Failure of these scripts is usually not very critical unless the UI is completely down or certain pages are not appearing as expected!

**#5)** We can have yet another set of tests that are simple but very laborious to be carried out manually. Tedious but simple tests are the ideal automation candidates, for example entering details of 1000 customers into the database has a simple functionality but extremely tedious to be carried out manually, such tests should be automated. If not, they mostly end up getting ignored and not tested.

### What NOT to Automate?

Given below are few tests which should not be automated.

#### #1) Negative tests/Failover tests

We should not attempt on automating [negative or failover tests](https://www.softwaretestinghelp.com/what-is-negative-testing/), as for these tests the testers need to think analytically and negative tests are not really straightforward to give a pass or fail result which can help us.

Negative tests will need a lot of manual intervention to simulate an actual disaster recovery kind of scenario. Just to exemplify we are testing features like web services reliability – to generalize it here the main aim of such tests would be to cause deliberate failures and see how well the product manages to be reliable.

Simulating the above failures are not straightforward, it can involve injecting some stubs or use some tools in between and automation is not the best way to go here.

#### #3) Tests with massive pre-setup

There are tests that require some enormous pre-requisites.

***For example,*** We may have a product that integrates with a 3rd party software for some of the functions, as product integrates with any messaging queue system which requires installation on a system, setting up of queues, creating queues etc.

The 3rd party software could be anything and the setup may be complex in nature and if such scripts are automated then these will forever be dependent on the function/setup of that 3rd party software.

**Simple Example of Test Automation**

When you are testing a software (on the web or desktop), you normally use a mouse and keyboard to perform your steps. Automation tool mimics those same steps by using scripting or a programming language.

**For Example**, if you are testing a calculator and the test case is that you have to add two numbers and see the result. The script will perform the same steps by making use of your mouse and keyboard.

The example is shown below.

**Manual Test Case Steps:**

1. Launch Calculator
2. Press 2
3. Press +
4. Press 3
5. Press =
6. The screen should display 5.
7. Close Calculator.

**Automation Script:**

|  |
| --- |
| //the example is written in MS Coded UI using c# language.  [TestMethod]  public void TestCalculator()  {    //launch the application  var app = ApplicationUnderTest.Launch("C:\\Windows\\System32\\calc.exe");    //do all the operations  Mouse.Click(button2);  Mouse.Click(buttonAdd);  Mouse.Click(button3);  Mouse.Click(buttonEqual);    //evaluate the results  Assert.AreEqual("5", txtResult.DisplayText,”Calculator is not showing 5);    //close the application  app.Close();  } |

The above script is just a duplication of your manual steps. The script is easy to create and easy to understand as well.

**What are Assertions?**

The second last line of the script needs some more explanation.

*Assert.AreEqual(“5”, txtResult.DisplayText,”Calculator is not showing 5);*

In every test case, we have some expected or predicted result at the end. In the above script, we have an expectation that “5” should be shown on the screen. The actual outcome is the result that is displayed on the screen. In every test case, we compare the expected outcome with the actual outcome.

Same goes for automation testing as well. The only difference here is, when we do that comparison in test automation, then it is called something else in every tool.

Some tools call it as “[Assertion](https://www.softwaretestinghelp.com/selenium-junit-framework-selenium-tutorial-11/)”, some call it as “[checkpoint](https://www.softwaretestinghelp.com/qtp-tutorial-14-bitmap-and-text-checkpoints/)” and some call it as “validation”. But basically, this is just a comparison. If this comparison fails, for **E.g.** a screen is showing 15 instead of 5 then this assertion/checkpoint/validation fails and your test case is marked as failed.

When a test case is failing due to an assertion then that means you have detected a bug through test automation. You must report it to your bug management system just like you normally do in manual testing.

In the above script, we have performed an assertion in the second last line. 5 is the expected outcome, *txtResult*. *DisplayText* is the actual outcome and if they are not equal, we will be shown a message that “Calculator is not showing 5”.

**There are some common “wrong” perceptions about automation.**

**They are:**

* We can automate every test case.
* Automating tests will reduce testing time enormously.
* No bugs are introduced if automation scripts are running smoothly.

We should be clear that automation can reduce testing time only for certain types of tests. Automating all the tests without any plan or sequence will lead to massive scripts which are heavy maintenance, fail often and need a lot of manual intervention too. Also, in constantly evolving products automation scripts may go obsolete and need some constant checks.

Grouping and automating the right candidates will save a whole lot of time and give all the benefits of automation.