**Automated Testing**

1. Decide what Test Cases to Automate
2. Test Early and Test Often
3. Select the Right Automated Testing Tool
4. Divide your Automated Testing Efforts
5. Create Good, Quality Test Data
6. Create Automated Tests that are Resistant to Changes in the UI

## Decide What Test Cases to Automate

It is impossible to automate all testing, so it is important to determine what test cases should be automated first.

The benefit of automated testing is linked to how many times a given test can be repeated. Tests that are only performed a few times are better left for manual testing. Good test cases for automation are ones that are run frequently and require large amounts of data to perform the same action.

You can get the most benefit out of your automated testing efforts by automating:

* Repetitive tests that run for multiple builds.
* Tests that tend to cause human error.
* Tests that require multiple data sets.
* Frequently used functionality that introduces high risk conditions.
* Tests that are impossible to perform manually.
* Tests that run on several different hardware or software platforms and configurations.
* Tests that take a lot of effort and time when manual testing.

Success in test automation requires careful planning and design work. Start out by creating an automation plan. This allows you to identify the initial set of tests to automate, and serve as a guide for future tests. First, you should define your goal for automated testing and determine which types of tests to automate. There are a few different types of testing, and each has its place in the testing process. For instance, unit testing is used to test a small part of the intended application. To test a certain piece of the application’s UI, you would use functional or GUI testing.

After determining your goal and which types of tests to automate, you should decide what actions your automated tests will perform. Don’t just create test steps that test various aspects of the application’s behavior at one time. Large, complex automated tests are difficult to edit and debug. It is best to divide your tests into several logical, smaller tests. It makes your test environment more coherent and manageable and allows you to share test code, test data and processes. You will get more opportunities to update your automated tests just by adding small tests that address new functionality. Test the functionality of your application as you add it, rather than waiting until the whole feature is implemented.

When creating tests, try to keep them small and focused on one objective. For example, separate tests for read-only versus read/write tests. This allows you to use these individual tests repeatedly without including them in every automated test.

Once you create several simple automated tests, you can group your tests into one, larger automated test. You can organize automated tests by the application’s functional area, major/minor division in the application, common functions or a base set of test data. If an automated test refers to other tests, you may need to create a test tree, where you can run tests in a specific order.

## Test Early and Test Often

To get the most out of your automated testing, testing should be started as early as possible and ran as often as needed. The earlier testers get involved in the life cycle of the project the better, and the more you test, the more bugs you find. Automated unit testing can be implemented on day one and then you can gradually build your automated test suite. Bugs detected early are a lot cheaper to fix than those discovered later in production or deployment.

With the shift left movement, developers and advanced testers are now empowered to build and run tests. Tools such as TestLeft allows users to run functional UI tests for web and desktop applications from within their favorite IDEs. With support for Visual Studio and Java IDEs such as IntelliJ and Eclipse, developers never have to leave the comfort of their ecosystem to validate application quality - meaning teams can quickly and easily shift left to deliver software faster.

### Start Shifting Left and Automate now with ****TestLeft****

## Select the Right Automated Testing Tool

Selecting an automated testing tool is essential for test automation. There are a lot of automated testing tools on the market, and it is important to choose the automated testing tool that best suits your overall requirements.

Consider these key points when selecting an automated testing tool:

* Support for your platforms and technology. Are you testing .Net, C# or WPF applications and on what operating systems? Are you going to test web applications? Do you need support for mobile application testing? Do you work with Android or iOS, or do you work with both operating systems?
* Flexibility for testers of all skill levels. Can your QA department write automated test scripts or is there a need for keyword testing?
* Feature rich but also easy to create automated tests. Does the automated testing tool support record-and-playback test creation as well as manual creation of automated tests; does it include features for implementing checkpoints to verify values, databases, or key functionality of your application?
* Create automated tests that are reusable, maintainable and resistant to changes in the applications UI. Will my automated tests break if my UI changes?

For detailed information about selecting automated testing tools for automated testing, see Selecting Automated Testing Tools.

## Divide Your Automated Testing Efforts

Usually, the creation of different tests is based on the QA engineers’ skill levels. It is important to identify the level of experience and skills for each of your team members and divide your automated testing efforts accordingly. For instance, writing automated test scripts requires expert knowledge of scripting languages. Thus, in order to perform these tasks, you should have QA engineers that know the script language provided by the automated testing tool.

Some team members may not be versed in writing automated test scripts. These QA engineers may be better at writing test cases. It is better when an automated testing tool has a way to create automated tests that do not require an in-depth knowledge of scripting languages, like TestComplete’s keyword tests feature. A keyword test (also known as keyword-driven testing) is a simple series of keywords with a specified action. With keyword tests, you can simulate keystrokes, click buttons, select menu items, call object methods and properties, and do a lot more. Keyword tests are often seen as an alternative to automated test scripts. Unlike scripts, they can be easily used by technical and non-technical users and allow users of all levels to create robust and powerful automated tests.

You should also collaborate on your automated testing project with other QA engineers in your department. Testing performed by a team is more effective for finding defects and the right automated testing tool allows you to share your projects with several testers.

## Create Good, Quality Test Data

Good test data is extremely useful for data-driven testing. The data that should be entered into input fields during an automated test is usually stored in an external file. This data might be read from a database or any other data source like text or XML files, Excel sheets, and database tables. A good automated testing tool actually understands the contents of the data files and iterates over the contents in the automated test. Using external data makes your automated tests reusable and easier to maintain. To add different testing scenarios, the data files can be easily extended with new data without needing to edit the actual automated test.

Typically, you create test data manually and then save it to the desired data storage. However, TestComplete provides you with the Data Generator that assists you in creating Table variables and Excel files that store test data. This approach lets you generate data of the desired type (integer numbers, strings, boolean values and so on) and automatically save this data to the specified variable or file. Using this feature, you decrease the time spent on preparing test data for data-driven tests. For more information on generating test data with TestComplete, see the Using Data Generators section in TestComplete’s help.

Creating test data for your automated tests is boring, but you should invest time and effort into creating data that is well structured. With good test data available, writing automated tests becomes a lot easier. The earlier you create good-quality data, the easier it is to extend existing automated tests along with the application's development.

## Create Automated Tests That Are Resistant to Changes in the UI

Automated tests created with scripts or keyword tests are dependent on the application under test. The user interface of the application may change between builds, especially in the early stages. These changes may affect the test results, or your automated tests may no longer work with future versions of the application. The problem is automated testing tools use a series of properties to identify and locate an object. Sometimes a testing tool relies on location coordinates to find the object. For instance, if the control caption or its location has changed, the automated test will no longer be able to find the object when it runs and will fail. To run the automated test successfully, you may need to replace old names with new ones in the entire project, before running the test against the new version of the application. However, if you provide unique names for your controls, it makes your automated tests resistant to these UI changes and ensures that your automated tests work without having to make changes to the test itself. This also eliminates the automated testing tool from relying on location coordinates to find the control, which is less stable and breaks easily.