Testing Types：

1. Functional GUI Testing:

Functional testing (or GUI testing) is testing of the application’s user interface that determines how the application and the user interact and whether the application performs properly. This typically includes how the application handles keyboard and mouse input and how it displays screen text, images, buttons, menus, dialog boxes, icons and toolbars.

1. Keyword-Driven Testing:

Keyword-Driven Testing is also known as table-driven testing or action word based testing, is a software testing methodology suitable for both manual and automated testing. This method separates the documentation of test cases-including the data to use-from the prescription of the way the test cases are executed. As a result, it separates the test creation process into two distinct stages: a design and development stage, and an execution stage.

1. Regression Testing:

Regression testing is any type of software testing that seeks to uncover new software bugs, or regressions, in existing functional and non-functional areas of a system after changes, such as enhancements, patches or configuration changes, have been made to them. The intent of regression testing is to ensure that a change such as those mentioned above has not introduced new faults.

Common methods of regression testing include rerunning previously-completed tests and checking whether program behaviors has changed and whether preciously-fixed faults have emerged.

1. Data-Driven Testing:

Data-Driven Testing is a term used in the testing of computer software to describe testing done using a table of conditions directly as test inputs and verifiable outputs as well as the process where test environment settings and control are not hard-coded. In the simplest form the tester supplies the inputs from a row in the table and expects the outputs which occur in the same row.

The table typically contains values which correspond to boundary or partition input spaces.

The advantage of Data-Driven testing is the ease to add additional inputs to the table when new partitions are discovered or added to the product or system under test.

It is the creation of test scripts to run together with their related data sets in a framework. The framework provides re-usable test logic to reduce maintenance and improve test coverage. Input and result data values can be stored in one or more central data sources or databased, the actual format and organization can be implementation specific.

Anything that has a potential to change (also called “variability”, and includes elements such as environment, end points, test data, locations, etc.) is separated out from the test logic (scripts) and moved into an “external asset”. This can be a configuration or test dataset. The logic executed in the script is dictated and by the data values.

Keyword-driven testing is similar except that the test case is contained in the set of data values and not embedded or “hard-coded” in the test script itself.

1. Unit Testing:

The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect. Each unit is tested separately before integrating them into modules to test the interfaces between modules. Unit testing has proven its value in that a large percentage of defects are identified during its use.

1. White-Box Testing:

It is also known as clear box testing, glass box testing, transparent box testing, and structural testing. It is a method of testing software that tests internal structures or workings of an application, as opposes to its functionality (i.e. black-box testing). The tester chooses inputs to exercise paths through the code and determine the appropriate outputs.

1. Distributed Testing:

It consists of two or more parts performed on separate computers that interact with each other. The crucial points of distributed test parts should be accumulated in a single common result set, while you should still keep the ability to analyze individual results of the needed test part.

1. Object-Driven Testing:

With object-driven testing, the entire test automation framework can be designed using nothing but objects, thus making test projects more flexible and easily maintainable than ordinary automated tests.

1. Coverage Testing:

There are a number of coverage criteria, the main ones being:

1. Function coverage – Has each function in the program been called?
2. Statement coverage – Has each node in the program been executed?
3. Decision coverage – Has every edge in the program been executed?
4. Condition coverage – Has each Boolean sub-expression evaluated both to true and false?
5. Condition/decision coverage – Both decision and condition coverage should be satisfied.
6. State coverage – Has each state in a finite-state machine been reached and explored?
7. Parameter Value coverage – In a method taking parameters, have all the common values for such parameters been considered?

Frameworks:

1. The Test Script Modularity Framework:

Requiring the creation of small, independent scripts that represent modules, sections, and functions of the application-under-test. These small scripts are then used in a hierarchical fashion to construct larger tests, realizing a particular test case.

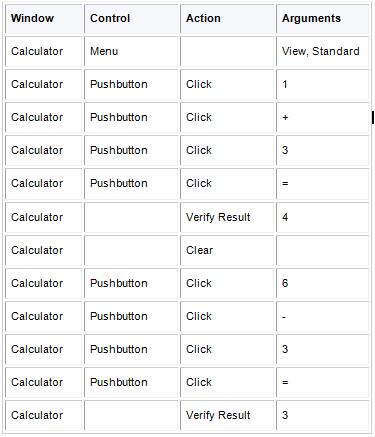
1. The Test Library Architecture Framework:

Very similar to Test Script Modularity Framework, but it divides the application-under-test into procedures and functions instead of scripts. This framework requires the creation of library files that represent modules, sections, and functions of the application-under-test. These library files are then called directly from the test case script.

1. They Keyword-Driven or Table-Driven Testing Framework:

This is an application-independent automation framework. It requires the development of data tables and keywords, independent of the test automation tool used to execute them and the test script code that “drives” the application-under-test and the data. In a keyword-driven test, the functionality of the application-under-test is documented in a table as well as in step-by-step instructions for each test.

This framework requires very little code to generate many test cases. The data tables are used to generate the individual test cases while the same code is reused.



1. The Data-Driven Testing Framework:

This is a framework where test input and output values are read from data files and are loaded into variables in captured or manually coded scripts. In this framework, variables are used for both input values and output verification values. Navigation through the program, reading of the data files, and logging of test status and information are all coded in the test script.

This is similar to table-driven testing in that the test case is contained in the data file and not in the script; the script is just a “driver”, or delivery mechanism for the data. Unlike in table-driven testing, though, the navigation data isn’t contained in the table structure. In data-driven testing, only test data is contained in the data files.

This framework tends to reduce the overall number of scripts you need in order to implement all of your test cases, and it offers the greatest flexibility when it comes to developing workarounds for bugs and performing maintenance. Much like table-driven testing, data-driven testing requires very little code to generate many test cases.

1. They Hybrid Test Automation Framework: