# F.I.R.S.T Principles of Unit Testing

Prepared By Senthilkumar K

## FAST, ISOLATED/INDEPENDENT, REPEATABLE, SELF-VALIDATING and THOROUGH/TIMELY

### Fast

•A developer should not hesitate to run the tests as they are slow.

•All of these including setup, the actual test and tear down should execute really fast (milliseconds) as you may have thousands of tests in your entire project.

### Isolated/Independent

•A test method should do the**3 As** => **Arrange, Act, Assert**

•Arrange: The data used in a test should not depend on the environment in which the test is running. All the data needed for a test should be arranged as part of the test.

•Act: Invoke the actual method under test.

•Assert: A test method should test for a single logical outcome, implying that typically there
should be only a single logical assert. A logical assert could have multiple physical asserts as
long as all the asserts test the state of a single object. In a few cases, an action can update
multiple objects.

•Avoid doing asserts in the Arrange part, let it throw exceptions and your test will still fail.

•No order-of-run dependency. They should pass or fail the same way in suite or when run individually.

•Do not do any more actions after the assert statement(s), preferably single logical assert.

## Repeatable

•A test method should NOT depend on any data in the environment/instance in which it is running.

•Deterministic results - should yield the same results every time and at every location where they run.
No dependency on date/time or random functions output.

•Each test should setup or arrange it's own data.
What if a set of tests need some common data? Use Data Helper classes that can setup this data for re-usability.

## Self-Validating

•No manual inspection required to check whether the test has passed or failed.

## Thorough

•Should cover every use case scenario and NOT just aim for 100% coverage.

•Tests for corner/edge/boundary values.

•Tests for large data sets - this will test runtime and space complexity.

•Tests for security with users having different roles - behaviour may be different based on user's role.

•Tests for large values - overflow and underflow errors for data types like integer.

•Tests for exceptions and errors.

•Tests for illegal arguments or bad inputs.