CI/CD
Lecture 9-Unit Testing

Omkarendra Tiwari

October 11, 2022

Testing

Why

- Proof of working code
- Ensure code keeps working
- Enable refactoring
- Documentation
- Release faster
- Feedbacks
- Code Coverage

Testing

Which one describes best

1. Process of demonstrating the absence of errors
2. To show that a program performs its functions correctly
3. A process of executing a program with intent to find errors

Testing

Psychology at play

- Human being tends to be highly goal oriented
- If the goal is to demonstrate that program is error free then...
- Same is applicable for otherwise

Another look

1. Process of demonstrating the absence of errors
2. To show that a program performs its functions correctly
3. A process of executing a program with intent to find errors

Testing

Economical Aspect

- White Box Testing
- Black Box Testing
  - Test on exhaustive set of valid inputs
  - Test on exhaustive set of invalid inputs
  - Example: Testing a compiler would require
    - Virtually infinite correct programs
    - Infinite incorrect programs

Testing

A few Principles/Guidelines

- Definition of the output is a necessary part of the test case
- A programmer/organization should avoid testing their own code
- Test case must be written for invalid/unexpected inputs as well as valid/expected inputs
- Test the program for if it does not do what it is supposed to do as well as it does what it is not supposed to do

Unit Testing

What

- Objective is to test the functionality of a piece of code

How

- For a fixed and verified Input, and fixed Output
- If Output is not known; Don't attempt to Unit Test
- Find a fixed Input and know the fixed Output for it
- For example, clustering problem.

Unit Testing: Best Practices

- Each Test tests exactly on thing
- In JUnit it is one test-method
- One assert per test-method

Unit Testing: Best Practices

- Write multiple tests rather than multiple asserts per test-method
- In multiple assert, if first fails, other asserts can't provide the information
- One model-class may be tested by multiple test-classes
- Tests must not be dependent on other tests

Unit Testing: Best Practices

Speed and Time

- Keep individual tests runtime in seconds
- Run quick tests first; fail early
- Long tests should be towards the end

Unit Testing: Best Practices

Bugs and Tests

- Tests should be focused
- For each bug exactly one test should fail
- If a test passed; Keep it quiet; Close to none output
- If a test failed; Output all hints/help that will make bug finding quick

Unit Tests

Input and Flakiness

- Rotate values for inputs such as 3, 8, 99, -11
- Avoid Input sources that are not-controlled, such as Network, File System (sometimes), system clock, gravity
- Multi-threaded, Math.random()

Unit Testing

Flakiness

- Different OS
- Floating point round off
- Integer width
- Path separator

Unit Testing: Write a failing Test

- When you find a bug
- When you are notified to the presence of a bug
- When you are going to refactor

Code Coverage

What

- A metric for assessing part of code covered under Tests

Kinds

- Function coverage: Percent of defined functions called
- Statement coverage: Statements executed
- Branches coverage: Branches executed
- Condition coverage: Boolean sub-expression tested for *true*/*false*
- Line coverage : Lines tested

TDD

Red-Green Refactor

- Write a test that fails; RED
- Write bare minimal code to make test pass
- Test again; GREEN