Effective Unit Testing

Motivators for

Sticking to Writing Better

Unit Tests

Why we write Unit Tests?

- Software needs to work
- Software needs to continue to work
- We need to be fast at producing software
- We are human, we make mistakes

Fundamental Principle of Unit Testing

- A piece of code that verifies that a known and fixed input produces a known and fixed output
- To create good tests
  - Use test cases where arguments about the output are not debatable
  - Don’t write tests for “fuzzy” input data
  - Use string literals. Avoid the use of domain constants (those are subject to changes)
  - Avoid access the network and, preferably, the file system

Write Your Test First

- Software development comes first
- “Test-First” helps you create better APIs
- Avoids having to know implementation details
- This practice/discipline helps you find test annotation errors (depending on the test suit)

Characteristics of Unit Tests

- Unit means “One.” Aim to test exactly one thing.
- Each test method is one test.
- Aim for a single `assert()` per test.
- Use Test Fixtures to share test setup logic. Avoid repetition.
- It is OK to have multiple test classes per domain class. Don’t feel pressured into having all your tests stuffed in a single test class. For example, if you have a domain class called `Foo`, avoid creating a single test class called `FooTest`.
- Test methods do not have to run in a pre-determined order.
- Tests, in some test runners, can run in parallel
- Don’t share data between tests (i.e. non-constant, mutable, static fields)

Speed of Tests

- A single tests should run in a second or less
- A complete suite should run in a minute or less
- Separate larger tests into additional suites
- Fail fast
- Developers tend to use fast test suites
- Developers avoid running slow test suites
- Tests that “**pass**” should not be verbose
- Tests that “**fail**” should be verbose
- Rotate your test data. Don’t use the same test value for multiple tests.

Dealing with Flakiness

- You have “flakiness” when a given test in a suite **passes**, but, sometimes, for some unknown reasons, the test **fails**.
- Work really hard to avoid this issue
- Sources of flakiness
  - Time dependencies
  - Network availability
  - Explicit randomness. Watch out when you see language-specific `rand()` functions in the test. Those could be sources of problems.
  - Multi-threading

System Skews – a different type of flakiness

- This occurs when tests run fine in your machine, but when your colleague runs the same suite, test fail for them
- Sources of System Skew Flakiness
  - Different OS
  - Threading issues
  - Other undefined behavior
    - Floating point roundoff
    - Integer width
    - Default character set (always use utf-8)
    - etc.

Logic Skews

- Avoid conditionals in your tests

```
if (x > 5) {
  assertTrue(y);
}
else {
 assertEquals(1, z);
}
```

Debugging

- Before fixing a bug, write a test that exposes it. If the test passes, then the bug isn't what you think it is
- This practice helps you augment your test suite
- This practice prevents the bug from getting re-introduced in the code base