Testing

Unit and Integration Testing

- Created by the developer, not a QA department
- Code to test your code

Unit Testing

- Unit tests are meant to test a single, small piece of code
- Often you have multiple unit tests for a single block of code to test many conditions
- Generally, when you think, “What happens if…” that can become a unit test.
- Unit tests are Isolated. They can be run independent of any external system (i.e., database)

Integration Tests

- Also meant to test a single piece of code.
- Not isolated, they do integrate with external systems.
  - Databases
  - File Systems
  - Web

Unit Testing Frameworks

- Define syntax for unit tests
- Provide the ability to run your unit tests and get back a pass, fail, and optionally a message
- Some common frameworks:
    - xUnit, JUnit for Java
    - NUnit for .NET
    - PHPUnit
    - Jasmine for JavaScript

Naming Best Practices

- Even though developers write these, they should be readable by a QA or product owner.
- Use long (seriously…really long) and explicit Names

```
MethodName_StateUnderTest_ExpectedBehavior

 IsAdult_AgeLessThan18_False

Should_ExpectedBehavior_When_StateUnderTest

 Should_ThrowException_When_AgeLessThan18
```

Unit Testing Conditions to Consider

- Null / Empty String
- 0, max value, min value, random value
- Expected exceptions get thrown
- Happy (expected) path

Test Driven Development (TDD)

- Development Process
  - Write a test (before you write your code)
  - Run the test (it will fail)
  - Make a small code change. The smallest code change that you can make that will result in your test passing.
  - Run the test
    - If it passes, add another test
    - If not, make a small code change

TDD - Why?

- When writing new classes it is important to understand how these classes will be used. By writing tests first, you can identify how you want to use and call into the classes that you are writing
- Self-Documenting code- shows future developers how your classes are intended to be used. This is important unless you want to be forever maintaining the same code base without moving onto something new

TDD - Why?

- Makes it easier to change the code in the future, whether for refactoring or bug fixes, you have a set of tests to run to verify that your code change did not have unintended consequences.
- By focusing on writing small and incremental chunks of code to just make a single test pass, theoretically you should be writing smaller more focused methods and classes which is just better design.
  - Much easier to find bugs in 2 lines of newly written code than 2000 lines.

TDD - Assertions

- https://phpunit.de/manual/current/en/appendixes.assertions.html
- https://phpunit.de/getting-started.html

Assertions

```
assertSame(var1, var2)
/*error if two variables do not have the same
type and value */

assertTrue(value)
/* error if value isn’t true */

assertInstanceOf(class, variable)
/* error if variable not instance of class */
```

Unit Testing and TDD Pitfalls

- Excessive indirection just to make a test pass.
    - TDD should simplify your code since you are writing code in testable blocks.
- Mocking can be helpful, but use your best judgement when writing many tests
    - Is it easier to use an integration test instead of a unit test?

More Pitfalls

- You should keep Assert statements to a minimum. Each unit test is meant to test a single condition.

- Don't test something just to write a test. It should help maintain the quality of the code. For example, a method that sets or gets a value is probably not very valuable to test. (when might it be?)