Module 1 - Lecture 14

Unit Testing

Review

- What is an abstract class?
- What is an abstract method?
- What are the differences between an abstract class and an interface?

Testing overview

- **Manual Testing** - a tester using the program as an end user would to determine if the program acts appropriately.

- **Automated Testing** - software that performs predefined actions and compares expected outcomes against actual outcomes.

Manual testing

Pros

- short term cost is lower
- more likely to find real user issues
- flexible

Cons

- higher long term cost
- much slower
- typically operates on an 8 hr schedule
- difficult/impossible to test code in isolation
- can be repetitive and not very stimulating
- can suffer from human error

Automated testing

Pros

- less expensive long term cost
- faster results
- more predictable

Cons

- higher short term cost
- can’t think for itself

Testing overview

- **Unit** - low level testing performed by programmers that validates individual units of code function as the programmer intended.
- **Integration** - validates the integration between units of code and outside dependencies such as a database or network resources.
- **End-to-end** - replicates users behavior using automation to verify user flows work as expected.
- **Acceptance** - validation performed from the perspective of a user of the system in order to verify that the functionality of the system satisfies user needs.

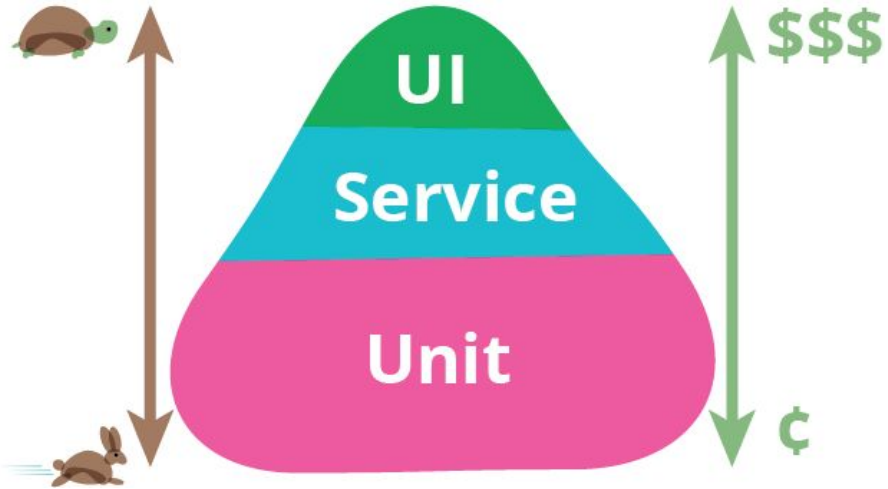
Testing overview

- **Exploratory** - explores the functionality of the system looking for defects, missing features, or other opportunities for improvement.

- **Regression** - validates the functionality of the system continues to operate as expected. Typically a subset of unit, integration, and/or end-to-end tests that are run after changes have been made to the system.

Testing overview

- **Unit -> Integration (Service) -> End-to-end (UI)**
- Runtime increases from left to right
- Maintenance and troubleshooting increases from left to right



Properties of a unit test

- **Fast** - the elapsed time of a unit test should be measured in milliseconds.
- **Reliable** / **Repeatable** - if a test passes/fails once, it should pass or fail every time, assuming the code hasn't changed.
- **Independent** - one test should not have an impact on another. A test should not require another test to run in order to succeed.
- **Obvious** - it should be easy to determine why a test failed.

Three part test

- **Arrange** the conditions of the test, such as setting up data.
- **Act** upon the action of interest i.e. the thing that we are testing.
- **Assert** that the expected outcome(s) occurred i.e. a certain value returned, a file exists, etc.

Unit Test Best Practices

- No external dependencies
- One *logical* assertion per test
- Test code is of the same quality as production code
- Test boundary cases
  - Empty arrays/lists, nulls, negative numbers
- At most one test class per class file

Code Coverage

Code coverage is the percentage of code which is covered by automated tests.

Code coverage measurement determines which statements in a body of code have been executed through a test run, and which statements have not.

We measure code coverage for the following reasons:

- To know how well our tests actually test our code
- To know whether we have enough testing in place
- To maintain the coverage over the lifecycle of a project

Let’s Code!

QUESTIONS?