Testing your code: Unit, Integration, System

The ultimate goal of tests

- reduce the probability of bugs in your code
- help you find the bugs in your tests
- lets you know if any underlying assumptions change

The method

- carefully controlled redundancy
- "double entry accounting"

Unit Tests vs System Tests vs. Integration Tests

Unit Tests

- Typically can be run entirely on your source code
- Typically requires no run-time component integration
- Purpose it to test the logic of an individual unit of code
- What is a unit?
  - Typically a class
  - Sometimes a package
- Purpose is both to prove correctness *and* to diagnose where bugs are found

Integration Test

- Typically designed to figure out if 2 systems play together nicely
- "Helps" you figure out what you missed in unit tests
- Typically more difficult that unit tests
  - Involves timing
  - May involve multi-process

System Test

- Typically focused on some kind of user-visible functionality
- Most difficult of the 3
- Typically different tools
  - Selenium

Terminology

- Test Fixture
  - the context for the test -- a setup
- Test Vector
  - A set of inputs for a test -- typically used for something which process data
- Test Case
  - the smallest runnable test
- Test Suite
  - A collection of tests

Coverage

- There is no such thing as complete coverage
  - line coverage
  - branch coverage
  - permutation of branch coverage
  - "coverage" python tool

Good Tests

- Should not depend on order of run (other tests)
- Should test one area or aspect fully
- Provides an example of how things are run
- Should cover all of the "primary" code
  - It's often difficult to cover every exception situation

Common Patterns

- Run all the unit tests during build
- “smoke” vs “full” tests

Example

- Unit Testing
- Mocks
- Coverage

Mocks

- Any class or service whose behavior can be controlled by the testing environment
  - Levels
    - Basic: Just hard code some test fixtures
    - Moderate: test fixtures are programatically controllable
    - Advanced: post-mort measurement of the interaction
- Mock Objects
- Mock Services

Designing for test

- Structure your code so that it can be tested
- No, this is not a waste of time. You're not just writting code, nor even correct code, but code which can be demonstrated to be correct

Other interesting tools

- sourceforge.net > Projects > Pyunitperf