**CS 362 – Module 6 Notes – Test Driven Development**

* **Test Driven Development** – the approach where one only writes new code if there exists at least one failing unit test (i.e. Someone writes unit tests *before* they have any code to test)
  + **TDD** is not a form of verification but primarily a way of approaching *implementation*.
  + **TDD** is a form of ***black box testing*** (i.e. you don’t see the code you’re trying to test for)
* **TDD Forces you to think about the requirements:**
  + TDD is often associated with the **Agile** software development method. It forces the devs to really think about the specifications and expected behavior for each feature. In order to write the unit tests first, the dev really has to think about appropriate *user stories* and *user cases*. This is because Agile isn’t as document and as requirement heavy as the **Waterfall Method** is, where every aspect and requirement of the software is documented and all directions must be followed.
* **TDD sets a good indicator of “done”:**
  + If your approach to unit testing is to write the code first, how do you know when you are done? For Agile development, where teams conduct *sprints* between frequent releases, knowing when the code meets the bare requirements helps to not waste time.
* **TDD reduces duplicate code:**
  + This trait is more geared towards TDD used with existing code bases and not so much “writing tests before any code is written”. The idea is to have devs write unit tests for the proposed new features/functionalities. Code should only be written if these new tests fail, otherwise it means the functionality already exists and there’s no need for new code. Helps keep multiple devs from implementing the same function and thus wasting time.
* **TDD Steps:**
  + **1.** Write a test
  + **2.** Run all currently written tests
    - if all tests pass, return to Step 1
    - If a test fails, proceed to Step 3
  + **3.** Write the bare minimum of code to make the test pass
  + **4.** Run all currently written tests
    - if all tests pass, return to Step 1
    - if the failing test is still failing, return to Step 3
  + **5.** Occasionally evaluate if the code can be refactored to reduce duplication or eliminate no longer used parts of the code
  + **6.** Eventually stop development after adding “enough” tests without triggering a new failure