Zachary Krausman

2/11/16

I’ve always been told that a computer does what you tell it to do, not what you want it to do. What this often means is that your program may run without crashing or giving error messages, but it isn’t giving the expected output. It’s often hard to figure out exactly where the error occurred. That’s why unit testing is important. There are so many small individual components to programs and it’s often hard to know when they’re working correctly. Unit testing allows testing of many cases. Test cases should be designed to find problems and should be designed as the program is being written not after it’s completed and you’re having trouble finding the problem. Also if testing is well done as the code develops, then new bugs are either from the new code or the new codes connection to the old code. Also unit testing is a quick way to see if code is working correctly for code that may be hard to run by its self, or code that requires user input or other resources. Trying to make sure that all cases are tested and all errors are caught will end up taking more time than writing the code. The idea isn’t to write perfect code, but to get indication that the code is working as expected especially with cases the code wasn’t written in mind for. The more things you have to do to run or compile executables, is just more steps that can go wrong. Try to make compiling and running code as simple of a process as possible. Keep a running list of all bugs, and don’t write new code until the bugs are fixed. It’s hard to understand what will need to be changed to fix bugs, so writing code that relies on other code which will change is a bad idea. You always want your code to be as bug free as possible so that if you need to do something with your code like release it or add new functionality, you can do that without bugs holding you back. If you’re working with other people, you need to write in a singular style. Often times companies will have a style they want you to adopt, but if not figure out a style that everyone will use. Regardless of how many people wrote however many different components of a program, the code should read the same throughout. Lastly, have both dedicated testers and randomly ask people to test your code. You want dedicated testers who know how to break code, so you can find errors. You want to randomly ask people to check your codes ease of use.

- What is TDD? Test Driven Development
- What is a continuous integration system? It's a practice of committing a working copy of all code at relatively frequent intervals. The idea is that multiple people are working on things that are going to need to work together, code needs to be continually tested to make sure the code is still working together correctly.
- What are the general guidelines for developing a good unit test?
    - Units should be tested, small individual components
    - Find cases that are designed to expose errors (Edge cases or exceptions that are handled)
    - Tests should only use code that has been used up to that point, future code will have its own tests. Don't use future code to check if past code is correct.