Pragmatic Unit Testing

Chapter 1.- Unit Testing

1.2 What is unit testing?

A unit test is a piece of code written by a developer that exercises a very small, specific area of functionality in the code being tested.

1.3 Why Should I Bother with Unit Testing?

Unit testing will make your life easier.

Beyond ensuring that the code does what you want, you need to ensure that the code does what you want all of the time, even when the winds are high, the parameters are suspect, the disk is full, and the network is sluggish.

1.5 How Do I Do Unit Testing?

The first step is to decide how to test the method in question— before writing the code itself. Next, you run the test itself. It's important that all the tests pass, not just the new one.

1.6 Excuses For Not Testing

- It takes too much time to write the tests
- My legacy code is impossible to test
- It's not my job to test my code
- But it compiles!
- I'm being paid to write code, not to write tests
- I feel guilty about putting testers and QA staff out of work
- My company won't let me run unit tests on the live system
- Yeah, we unit test already

Chapter 4.- What to test: The Right BICEP

Six specific areas to test that will help strengthen your testing skills, using your Right-BICEP:

- Right – Are the results right?
    - See if the expected results are right
    - Key question: "if the code ran correctly, how would I know?"
    - Data files are useful if you have to test codes with large amounts of data. Put the values in a separated file that the unit test reads in.
    -
- B – Are all the boundary conditions CORRECT?
    - Boundary conditions for data

        - Inconsistent input values
        - Badly formatted data.
        - Empty or missing values
        - Values far in excess of reasonable expectations
        - Duplicates in lists that shouldn’t have duplicates.
        - Ordered lists that aren’t and vice-versa.
        - Things that arrive out of order
    - Easy way to think boundary conditions: “CORRECT”
        - Conformance
        - Ordering
        - Range
        - Reference
        - Existence
        - Cardinality
        - Time (absolute and relative)
- I — Can you check inverse relationships?
    - Build a method that checks that some data was successfully inserted into a database.
- C — Can you cross-check results using other means?
    - Check for the algorithm that best calculates some quantity
- E — Can you force error conditions to happen?
    - Consider what kinds of errors or other environmental constraints you might introduce to test your method.
- P — Are performance characteristics within bounds?

Chapter 5.- CORRECT Boundary Conditions

- Conformance
    - Validating formatted string data such as email-addresses, phone numbers, account numbers, or file names.
- Ordering
    - Put information in order
- Range
    - If you input a number that is out of range, your code must specify this error.
- Reference
    - Stablish conditions to run a specific part of the code or the code itself
- Existence
    - Write specific exceptions in your code in order to detect errors easily.
- Cardinality
    - Think about ways to test how well your method counts, and check to see how many of a thing you may have. (Fence Post problem)
- Time
    - Relative time (ordering in time)
    - Absolute time (elapsed and wall clock)

  - Concurrency issues

Testing your code

Some general rules of testing:

- Use long and descriptive names for testing functions. The style guide here is slightly different than that of running code, where short names are often preferred. The reason is testing functions are never called explicitly. `square()` or even `sqr()` is ok in running code, but in testing code you would have names such as `test_square_of_number_2()`, `test_square_negative_number()`. These function names are displayed when a test fails, and should be as descriptive as possible.
- A testing unit should focus on one tiny bit of functionality and prove it correct.
- Always run the full test suite before a coding session, and run it again after. This will give you more confidence that you did not break anything in the rest of the code.