CS126 - Testing

PrairieLearn - get it done!

Don’t panic about “low code review scores”.
Why vs How

Knowing “why” motivates understanding of “how” and “what”. When you know why, you seek how. Once you have gained this understanding, the rest of the pieces fall into place. This WILL lead to you having an easier time meeting expectations (your grade). It is also what distinguishes developers from one another.
Reading
(understanding why)

PrairieLearn (Testing)
https://courses.grainger.illinois.edu/cs126/sp2021/assignments/

Software Testing Stuff (types of testing)
http://www.softwaretestingstuff.com/2008/12/software-testing-techniques-and-levels.html

Unit Testing: You’re doing it wrong
https://medium.com/@Cyrdup/unit-testing-youre-doing-it-wrong-407a07692989
Testing Misconceptions

Unit Testing: You’re doing it wrong

https://medium.com/@Cyrdup/unit-testing-youre-doing-it-wrong-407a07692989

1. TDD is all about unit tests.

2. Automated testing is all about unit tests

3. 100% code coverage requires extensive unit testing
Testing Misconceptions

4. You have to make private methods public to reach 100% coverage

Again, no! private methods will be tested through public entry points. Once again, unit testing is not about testing every method in isolation.

5. Some code does not need be tested

6. You need to use a mocking framework
Testing Misconceptions

7. Tests are expensive to write

8. The ‘testing pyramid’ is the ultimate testing strategy
What about some truths?

1. Unit tests are not about testing a method in isolation

3. There is a tooling problem

4. It is difficult
What about some truths?

5. Tests require maintenance

To sum it up: **tests are part of your codebase and must be treated as such.** Which leads to the next truth:
What about some truths?

6. Having too many tests is a problem

7. Throwing away tests is a hygienic move
What about some truths?

8. Automated tests are useful

An interesting and important 2014 study analyzed 198 user reported issues on distributed systems (incl HBase, Cassandra, Zookeeper). Among several important findings, it concluded that 77% of the analyzed production issues could have been reproduced by a unit test.

Another key finding was that almost all catastrophic failures were the result of incorrect error handling.
Informal Testing: Ad hoc testing performed without a documented set of objectives or plans. Informal testing relies on the intuition and skills of the individual performing the testing. Experienced engineers can be productive in this mode by mentally performing test cases for the scenarios being exercised.

This is also what you do when you test your code as you develop it. Do not think of tests only in the formal sense. That is, do not only think of your test suite for your grade. Those are the formal tests that you would use to verify that your code exhibits the behaviors in your specification.
Code Style and Extensibility

(How do I get the best score?)

Ease of modification:

1. recompiling
2. startup configuration
3. runtime configuration (user selection)

- Each of these considerations affects the exact type of implementation you will need to choose.
- Which method you use depends on what the goals of your application are.
What’s wrong with this?

String Str1 = new String("This is my board");
If (str1.length <> 9) {
   // do something useful
}

final integer stringLength = 9;
What is expected of us in CS126 in regards to testing? (How do I get the best score?)

- As always, we expect you to think about what you are doing and what the limitations are of your approach.
- You should endeavor to write meaningful tests and use them to drive your design.
- Once you have understood the responsibilities and limitations of testing, you will be expected to exhibit that understanding on all subsequent assignments.
  - Meaningful test cases
  - No redundancy
  - No obvious “holes”

Java and Testing <slides>
Difficulties
Realistically, what should we expect?

1. it forces you to think problem-first, while solution-first is probably your comfort zone
2. it constrains your design
3. it gives you the impression of being unproductive. The truth is quite the opposite.
(Writing tests IS productive)
Benefits

1. Problem-first is the right focus!
2. Constraints help you drive the design. If you have adopted problem-first, this should drive good design.
3. Worst case, tests will be thrown away. But they helped you build a solution and deeper understanding of the problem.
4. At best, they prevent future regression, and provide help and documentation for future developers.

One misconception: tests ensure correctness. They do not. Your tests document behaviors that you have verified and some assumptions that you have made about those behaviors.
Limitations

- Unit tests only test “visible” methods.
- It is challenging to write code that thoroughly tests every piece of code. That is, to get good coverage.
- It may not help you as you are composing code.
  - How do I test little pieces of functionality that are only parts of the outward facing behaviors?
  - How do I test private/helper methods?

Demo Notes

TicTacToeTest is just a specialized Main
Let’s look at my tests.
  They pass so my code works, right?
**Connect Four**

Let’s look at a similar game


This is essentially tic-tac-toe but
- has a larger board
- Takes 4 to win
- Moves are limited (add checker to column)

Create test examples

`assertEquals` - [http://junit.sourceforge.net/javadoc/org/junit/Assert.html](http://junit.sourceforge.net/javadoc/org/junit/Assert.html)

“@Test”

Success does not ensure correctness

Run Configuration

% coverage

Test Cases - minimum number of tests
Testing Private Methods

Do not make private methods public to test them - they are PART of public functionality (behavior)

- In my example, “BadColumn” is ACTUALLY a type of UnreachableState
- I can do this temporarily to test a private method. HOWEVER, you should probably avoid this type of practice.
- There are other ways! (I have used the term “informal testing” for this)
  - Print interim results and confirm by hand
  - Create output methods in your class for testing purposes.
  - Use the debugger and step through your code

Let’s examine some artifacts of “copy/paste coding”
  - We forget to “customize” things
  - You will not get a free pass on this