Testing

CMSC 202

Overview

- What is software testing?
- What is unit testing?
- Why/when to test?
- What makes a good test?
- What to test?

What is Software Testing?

- Software testing is any activity aimed at evaluating an attribute or capability of a program or system and determining ***that it meets its required results***.

— William Hetzel
"The Complete Guide to Software Testing"

Types of Software Testing

- Unit Testing
  - Verifies the functionality of a specific chunk of code, usually at the function/class level
- Integration Testing
  - Testing of combined modules as a whole
- System Testing
  - Tests fully integrated system against requirements
- System Integration Testing
  - Testing between multiple systems

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.

- Usually a unit test exercises some particular method in a particular context.

— Andy Hunt & Dave Thomas
"Pragmatic Unit Testing"

Unit Testing

- Also known as ***component testing***
- In OOP, typically ensures that a method/class works as designed
- Written by *developers* to test their code.
  - Also known as ***white box testing***

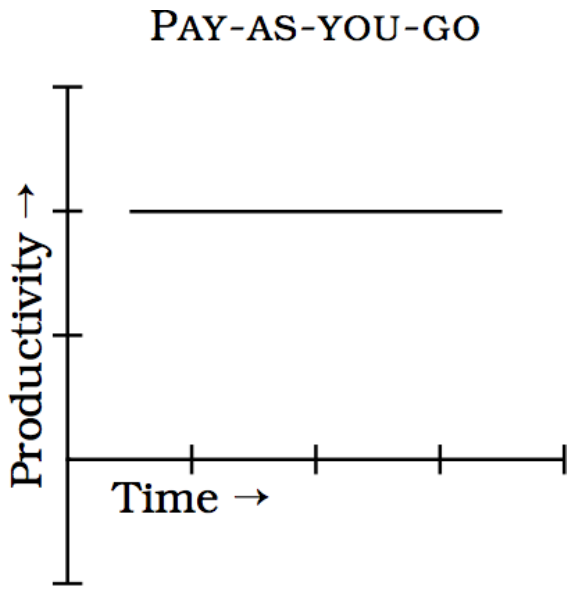
Why Test?

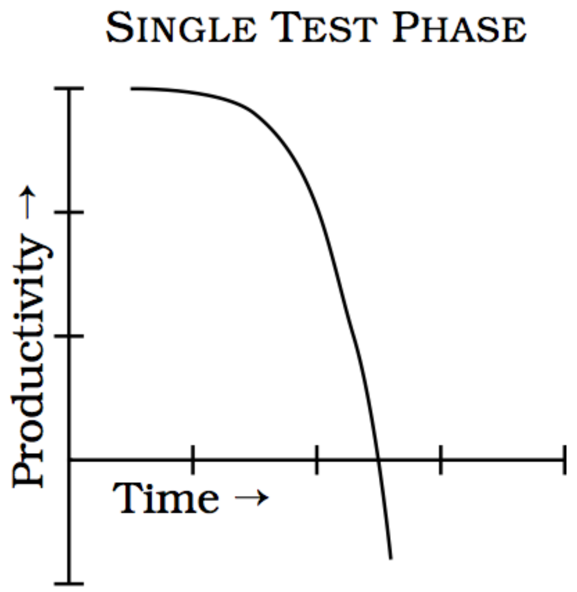
- You wouldn’t do this without a safety net.
- Why develop your code without one?

When to Test

- How many of you write almost all of your code and then write some tests …
  - To fulfill project requirements?
  - To exercise and test your code?
- How many of you incrementally write tests to exercise code as your write it?
- Anyone write the tests first?

Pay Now or Pay Later





- It’s cheaper in the long run to “pay as you go.”
- Minimizes trying to solve many problems at once at the end of your development cycle

Test Driven Development

- Test Driven Development (TDD) takes this “pay early” approach a step further by requiring that you write the tests before writing non-test code.
    1. Add tests
    2. Run tests, new tests should fail
    3. Write code to satisfy tests
    4. Re-run tests; all tests should pass
    5. Refactor as needed
    6. Repeat

Properties of Good Unit Tests

- What are things we aim for in good tests?
  - Repeatable
    - Should be able to be re-run producing the same results (avoid randomness, getting current time, etc.)
  - Independent
    - Only test one feature (method) at a time.
    - Tests should not be dependent upon one another.
  - Provide value
    - Testing simple getters/setters is probably not a good use of time.
  - Thorough
    - Test all class invariants, pre/post conditions, edge cases.

Thoroughness

- In order for your tests to be thorough, you need to check for several things.
  - General Correctness
  - Boundary Conditions
  - Error Conditions

General Correctness

- These are the so-called easy tests to write.
- These test the “general” cases.

Boundary Conditions

- Ordering
  - Does various ordering affect the outcome?
- Range
  - zero, minimum, maximum, positive #s, negative #s
- Existence
  - Null values for reference parameters
  - Empty things
    - Collections (e.g. arrays)
    - Strings
- Cardinality
  - Expected number of items

Error Conditions

- Are the right exceptions getting raised under the right conditions?
- I/O issues
  - Missing files
  - Unreadable files
  - Empty files

Exercise

- Identify test cases for the following method.

```
public static int largest(int[ ] list) {
    /* code */
}
```

- What tests might we have for each of the following areas?
  - – General correctness
  - – Boundary conditions
  - – Error conditions

A Buggy Implementation

- How many of your tests failed on the following buggy implementation of largest?

```
public static int largest(int[ ] list) {
        int max = Integer.MAX_VALUE;
        for(int i = 0; i < list.length - 1; i++) {
                if(list[i] > max) {
                        max = list[i];
                }
        }
        return max;
}
```

A Much Improved largest Method

```
public static int largest(int[] list) {
        if(list == null) {
                throw new IllegalArgumentException("list cannot be null");
        } else if (list.length == 0) {
                throw new IllegalArgumentException("list cannot be empty");
        }

        int max = Integer.MIN_VALUE;
        for(int i = 0; i < list.length; i++) {
                if(list[i] > max) {
                        max = list[i];
                }
        }
        return max;
}
```

Additional Resources

- Pragmatic Unit Testing in Java with JUnit
  - Free Introduction chapter
  - Free testing Summary cheat-sheet
- JUnit Test Infected: Programmers Love Writing Tests