Lecture 5b: Testing

CS 70: Data Structures and Program Development
Thursday, February 20, 2020

Learning Goals

- **I can explain what testing is.**
- **I can explain why testing is useful.**

Testing: Philosophy of Science Point of View

"My proposal is based on an asymmetry between verifiability and falsifiability; an asymmetry which results from the logical form of universal statements. For these are never derivable from singular statements, but can be contradicted by singular statements."

- Karl Popper: The Logic of Scientific Discovery 1959

"Program testing can be used to show the presence of bugs, but never to show their absence!""

- Edsger Dijkstra. Notes On Structured Programming 1970

What does that have to do with testing?

- **What is the purpose of testing?**
  - To show that the code has no bugs? A nice ideal.
  - Is exhaustive testing possible?
  - Consider an application with input fields:
    - First Name: up to 20 characters
    - Last Name: up to 20 characters
    - Phone Number: 10 digits
- **The goal of testing is to *find errors*.**

Why test during development?

- **Depends on the project.**
  - Not necessary in ALL cases
    - small, simple pieces of code
    - code that won't ever be re-used
  - Designing an interface for many other users? definitely test.
  - In projects where one makes big design choices or there are multiple strategies, testing is especially important.
- **Aspirationally: tests come first.**

Test first?

Test first gives you a hint as to whether your interface suffices.

- Is it enough to solve a problem?

Test first suggests that your interface is even testable.

- Does your code actually have the *potential* to work?

Test first increases your velocity in general.

- You get to your solution quicker
- Get to see how far you have gotten and how much more you have until you reach your goal

What is a test?

A test should have several components:

- The test input or scenario
- The expected result
- Documentation
  - What part of the requirements is being tested?
  - What is the reasoning behind the test?

How do we test?

- Find a framework to make testing easy
  - i.e. the lightweight testing-logger we provided
  - affirm test conditions
- Testing Domain Specific Languages (DSLs)
- C++: gtest (an open source project, stands for Google Test).
- Python: pytest
- Java: junit, Truth

Equivalence Partitioning

What are some good test values for this function?

```
double compute(double x){
  double y = x + 1.0;
  if(y < 10.0){
    return y;
  }
  return 10;
}
```


Boundary Analysis

- **Extremely common mistakes:**
  - you write *num* when you meant *num - 1*
  - you write >= when you meant >
- **Explicitly write tests to discover these types of errors**

- **Input values to test?**

```
double compute(double x){
  double y = x + 1.0;
  if(y < 10.0){
    return y;
  }
  return 10;
}
```


Explicitly test for bad data

Examples:

- Too little data:
  - an empty vector
- Too much data:
  - array of 1 million employees
- Invalid data:
  - Negative student ID number

What about uninitialized data??

```
int x;
compute(x);
```


Explicitly test for good data

Examples:

- Nominal data: middle of the road, expected data
  e.g. 10 employees in database
- Minimum nominal configuration
  e.g. 1 employee in database
- Maximum nominal configuration
  e.g. 1,000 employees in database

Unit testing

Test individual units of code:

- typically, test every single function

How much testing is enough? At a *minimum*

- Cover every statement
- Cover every branch

Integration testing

After individual units (e.g. functions) are tested, do they correctly work together?

Two main ways to combine units:

Output of one passed to the other

```
int x = function1(y);
int z = function2(x);
```

One function is called by another

```
int function3(int x){
  int y = function4(x)
  return y;
}
```


Integration testing with a Driver

For integration testing, a driver is a function or class whose sole purpose is to combine two or more units.

```
int driver(int y){
  x = function1(y);
  z = function2(x);
  return z;
}
```


Exercise: Brainstorm test inputs

Consider an application with input fields:

```
* First Name: up to 20 characters
* Last Name: up to 20 characters
* Phone Number: 10 digits
```

Come up with some test inputs