CISC 3120

C07: Testing and JUnit

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Recap and issues
  - Grades and feedback
  - Assignments & projects
  - Review guide & CodeLabs
- Testing
- Junit
- Assignments
  - Practices

Recap and Issues

- Any questions?
  - Grades and feedback
  - Assignments & projects
  - Review guide & CodeLabs

Software Failures

- Ariane 5 rocket explosion (June 4, 1996)
    - "Overflow from conversion from a 64-bit floating point number to a 16-bit signed integer value …"
- Therac-25 lethal radiation overdose (June 1985 ~ Jan 1987)
    - "Some basic software engineering principles were apparently violated …"
- Mars Climate Orbiter disintegration (December 11, 1998)
    - "... In the case of the ground software, … was in English units of pounds (force)-seconds (lbf-s) rather than the metric units specified …"
- FBI Virtual Case File project abandonment (2000 ~ 2005)
    - " …a systematic failure of software engineering practices …"

Recent Stories in the Airlines Industry

- “The big computer systems that get airplanes, passengers and baggage to their destinations every day are having a bad summer.” (NY Times, August 8, 2016)
  - 1,000 0f 6,000 Delta flights canceled, August 8-9, 2016
  - 2,300 canceled flights over 4 days at Southwest Airlines, ~July 22, 2016
  - Hundreds of flights grounded at United Airlines, July 2015
  - An iPad software glitch caused two days of problems for American Airlines, the airline said Wednesday …

Software Quality Assurance

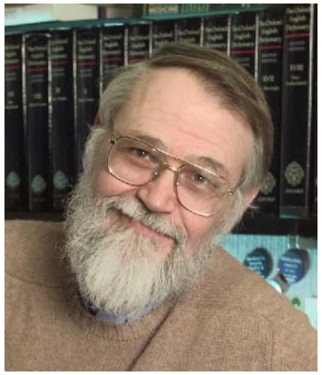
- Verification
  - Did you build the thing right? (Did you meet the specification?)
- Validation
  - Did you build the right thing? (Is this what the customer wants? That is, is the specification correct?)
- Two approaches
  - Testing
  - Formal methods

Formal Methods for Software Verification

- Start with a formal specification and prove code behavior follows that of specification
  - Mathematical proofs
  - Humans do the proofs
  - Computers do the proofs
    - Automatic theorem proving
    - Model checking
  - In practice, mostly done for hardware; done in very limited cases for software
    - Computational intensive: hard to test, not too large spec., error repair cost prohibitive, critical components or systems

"In 1981, Edmund M. Clarke and E. Allen Emerson, working in the USA, and Joseph Sifakis working independently in France, authored seminal papers that founded what has become the highly successful field of Model Checking. This verification technology provides an algorithmic means of determining whether an abstract model--representing, for example, a hardware or software design--satisfies a formal specification expressed as a temporal logic formula. Moreover, if the property does not hold, the method identifies a counterexample execution that shows the source of the problem. The progression of Model Checking to the point where it can be successfully used for complex systems has required the development of sophisticated means of coping with what is known as the state explosion problem. Great strides have been made on this problem over the past 27 years by what is now a very large international research community. As a result many major hardware and software companies are now using Model Checking in practice. Examples of its use include the verification of VLSI circuits, communication protocols, software device drivers, real-time embedded systems, and security algorithms." -- by ACM, 2007

Testing for Software Verification



- Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it? -- Brian Kernighan

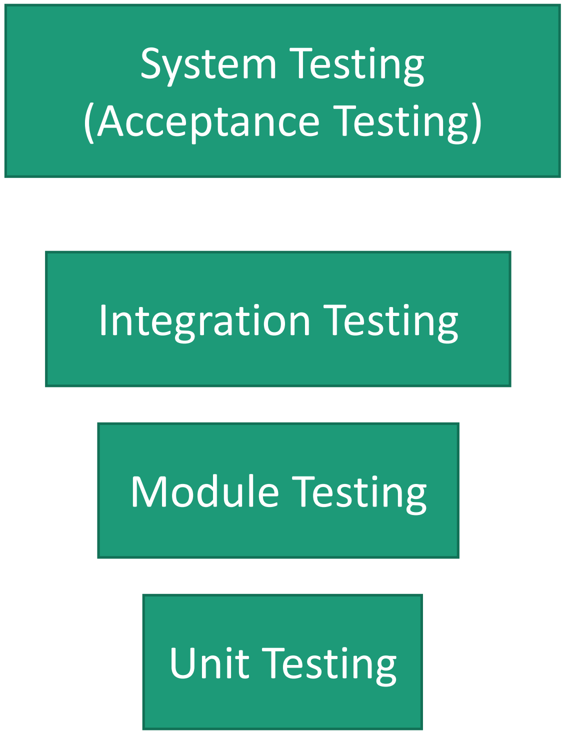
- Programming testing can be used to show the presence of bugs, but never to show their absence! – Edsger W. Dijkstra

Testing

- Exhaustive testing is infeasible
  - e.g., 1 nanosecond for test a program that has one 64-bit input, how long does it take to test all possible input values?
- Reduce the space
  - Perform different tests at different phases of software development

Different Types of Tests

- System testing (acceptance testing): if the integrated program meets its specification
- Integration testing: interfaces between units have consistent assumption and communicate correctly
- Module testing: tests across individual units (e.g., across classes)
- Unit testing: single method does what was expected (e.g., within a single class)



Perspectives: Black-box vs. White-box Tests

- Black-box tests
  - Test design is solely based on the program's external specifications
- White-box (glass-box) tests
  - Test design reflects knowledge about the program's implementation, e.g., developers' doing the tests
- For it or against it?

Test Coverage

- The fraction of the possible program execution paths that have been tested
  - 100% of test coverage is no guarantee of design reliability
  - Quality of tests also matters

Common Test Coverage Levels

- S0 (method coverage)
  - Is every method executed at least once by the test suite?
- S1 (call coverage or entry/exit coverage)
  - Has each method been called from every place it can be called?
- C0 (statement coverage)
  - Is every statement of the source code executed at least once by the test suite?
- C1 (branch coverage)
  - Has each branch been taken in each direction at least once?
- C2 (path coverage)
  - Has every possible route through the code been executed?

Sample Code to Test

```
public class MyClass {
    public void foo(boolean x, boolean y, boolean z) {
       if (x)
           if (y && z) bar(0);
           else
               bar(1);
    }
  public boolean bar(x) {
      return x;
    }
}
```

Test Coverage S0

```
public class MyClass {
    public void foo(boolean x,
boolean y, boolean z) {
        if (x)
            if (y && z) bar(0);
            else
                bar(1);
    }
  public boolean bar(x) {
      return x;
  }
}
```

Satisfying S0 requiring calling foo and bar at least once each in the tests

Test Coverage S1

```
1.   public class MyClass {
2.      public void foo(boolean x,
     boolean y, boolean z) {
3.          if (x)
4.              if (y && z) bar(0);
5.              else
6.                  bar(1);
7.     }
8.    public boolean bar(x) {
9.        return x;
10.    }
11.  }
```

Satisfying S1 requiring calling bar from both line 4 and line 6 in the test suites

Test Coverage C0

```
public class MyClass {
    public void foo(boolean x, boolean y, boolean z) {
        if (x)
            if (y && z) bar(0);
            else
                bar(1);
    }
  public boolean bar(x) {
      return x;
    }
}
```

Counting both branches of a conditional as a single statement, satisfying C0 requiring calling foo at least once with x true, and at least once with y false

Test Coverage C1

```
public class MyClass {
    public void foo(boolean x, boolean y, boolean z) {
        if (x)
            if (y && z) bar(0);
            else
                bar(1);
    }
  public boolean bar(x) {
      return x;
  }
}
```

Satisfying C1 requiring calling foo at least once with x true, and with x false, and with y && z true and false.

Test Coverage C2

```
public class MyClass {
  public void foo(boolean x, boolean y, boolean z) {
     if (x)
        if (y && z) bar(0);
        else
           bar(1);
  }
  public boolean bar(x) {
     return x;
  }
}
```

Satisfying C2 requiring calling foo with all 8 combinations of values of x, y, and z

Modified Condition/Decision Coverage (MCDC)

- Combines a subset of the above levels
  - Each point of entry and exit in the program have been invoked at least once
  - Every decision in the code has taken all possible outcomes at least once
  - Each condition in a decision has been shown to independently affect that decision's outcome

Achieving Test Coverage

- 100% of C0 coverage is not unreasonable.
- Achieving C1 coverage requires careful construction of tests.
- C2 is the most difficult of all, and the additional value of 100% of C2 is debatable.

Questions?

- Examples of catastrophic software failures
- Software quality assurance
- A few important concepts on software testing

Team Discussion & Exercise

- Design tests for a method

Unit and Functional Testing

- Recall …
  - Unit testing: single method does what was expected (e.g., within a single class)
  - Functional testing: a well-defined subset of the code does what was expected (e.g., several methods and classes)
- Tests
  - Calls to methods with different input parameters
  - Asserts on the effects of method calls
  - Aims for high coverage
  - Almost always white-box, and performed by developers

Test Assertion

- An expression encapsulates some testable logic about a target under test

JUnit

- A unit testing framework for Java
- Test assertion in JUnit
  - It throws an exception if it evaluates to false

Unit Test Example with JUnit

- Some of you completed the Array and ArrayList assignment
- Let’s use it as an example
  - Test whether the “delete” method functions as specified.

```
public class FruitArray
{ …
   public void delete(String fruitName) { … }
…
}
```

What is the Specification?

```
public class FruitArray
{ …
   public void delete(String fruitName) { … }
…
}
```

What is the Specification?

- Remove an element, “shrink” the “list”, as if the element had never been in the “list”.

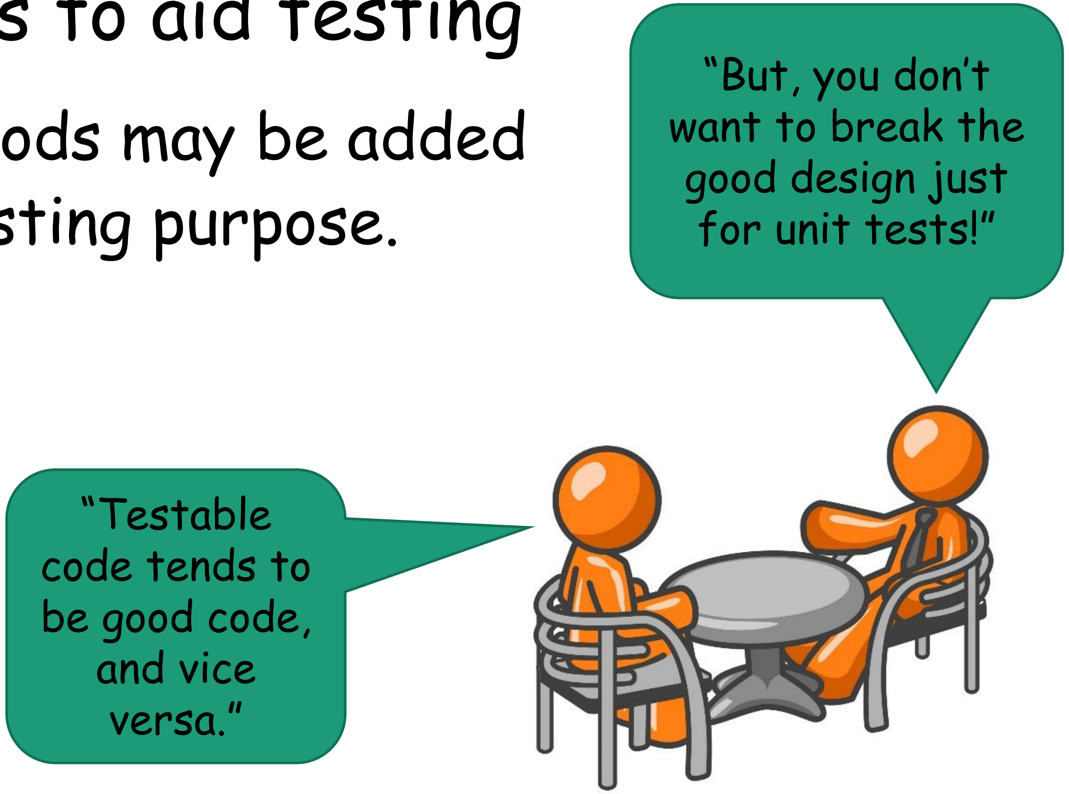
```
public class FruitArray
{ …
   public void delete(String fruitName) { … }
…
}
```

What is the Test Assertion?

- It is expected that upon an item is deleted, the object should be identical to another object of the class that does not have the item from beginning with, but has all the other items.

Example Design

- Add methods to aid testing
  - Some methods may be added just for testing purpose.


"But, you don't want to break the good design just for unit tests!"
"Testable code tends to be good code, and vice versa."

Example Implementation in JUnit 4

- Use JUnit 4, assume Maven project in Eclipse
- A few steps
  - Update pom.xml

```
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

  - Remove AppTest.Java
  - Create your own test class (See https://github.com/junit-team/junit4/wiki/getting-started)
  - Complete the example in "ArrayArrayList" in the "sampleprograms" repository

FruitArrayTest.java

```java
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertArrayEquals;
import org.junit.Test;
public class FruitArrayTest
{
  @Test // Important: annotate a test
  public void testDelete()
  {
    testDeleteByDeleting1st();
    testDeleteByDeleting3rd();
    testDeleteByDeletingLast();
    testDeleteIgnoreCaseByDeleting1st();
    testDeleteIgnoreCaseByDeletingLast();
  }
```

```java
private void testDeleteByDeleting1st() {
    String[] fruits = {new String("Apple"),
                       new String("Banana"),
                       new String("Kiwi"),
                       new String("Mango"),
                       new String("Orange")};
    FruitArray fruitArray = new FruitArray(fruits);
    fruitArray.delete(new String("Apple"));
    assertEquals(fruitArray.getSize(), fruits.length - 1);
    assertEquals(fruitArray.getCapacity(), fruits.length);
    assertArrayEquals(fruits, fruitArray.getFruitsAsArray());
  }
```

Note: the purpose of the test is to show how you write tests in a JUnit. This test is actually poorly designed. See the sample code in the repo for better ones.

Did We Pass the Tests?

Questions

- JUnit
- Examples in JUnit 4
    - Update pom.xml
    - Use the sample code as the template or the one in the JUnit 4 wiki page

Assignments

- How are we doing?
  - Projects
  - Assignments
- Review questions and CodeLab