

Unit Testing, for Better and Faster Code Development

By Samer Zein, PhD

Software QA, CS Dept., Birzeit University - Dr. Samer Zein

Introduction



- Many organizations have grand intentions when it comes to testing,
- but tend to test only toward the end of a project, when the mounting schedule pressures cause testing to be curtailed, or eliminated entirely.
- And they get to ship the product to customer!!

Software QA, CS Dept., Birzeit University - Dr. Samer Zein

all resources of this lecture are taken from "Pragmatic Unit Testing in Java" Book
STUDENTS-HUB.com Uploaded By: anonymous

Introduction..2

- Many programmers feel that testing is just a nuisance: an unwanted bother that merely distracts from the real business at hand-cutting code.
- Everyone agrees that more testing is needed, in the same way that everyone agrees you should
 - eat your broccoli,
 - stop smoking, get plenty of rest,
 - and exercise regularly.
- That doesn't mean that any of us actually do these things, however.

Introduction..3

- Unit testing is much more than these!
- It is like an awesome sauce that makes everything taste better.
- Unit testing isn't for end users, managers or team leaders;
- It is done by programmers, for the programmers.!!
- Unit testing can mean the difference between your success and your failure

So what is Unit Testing

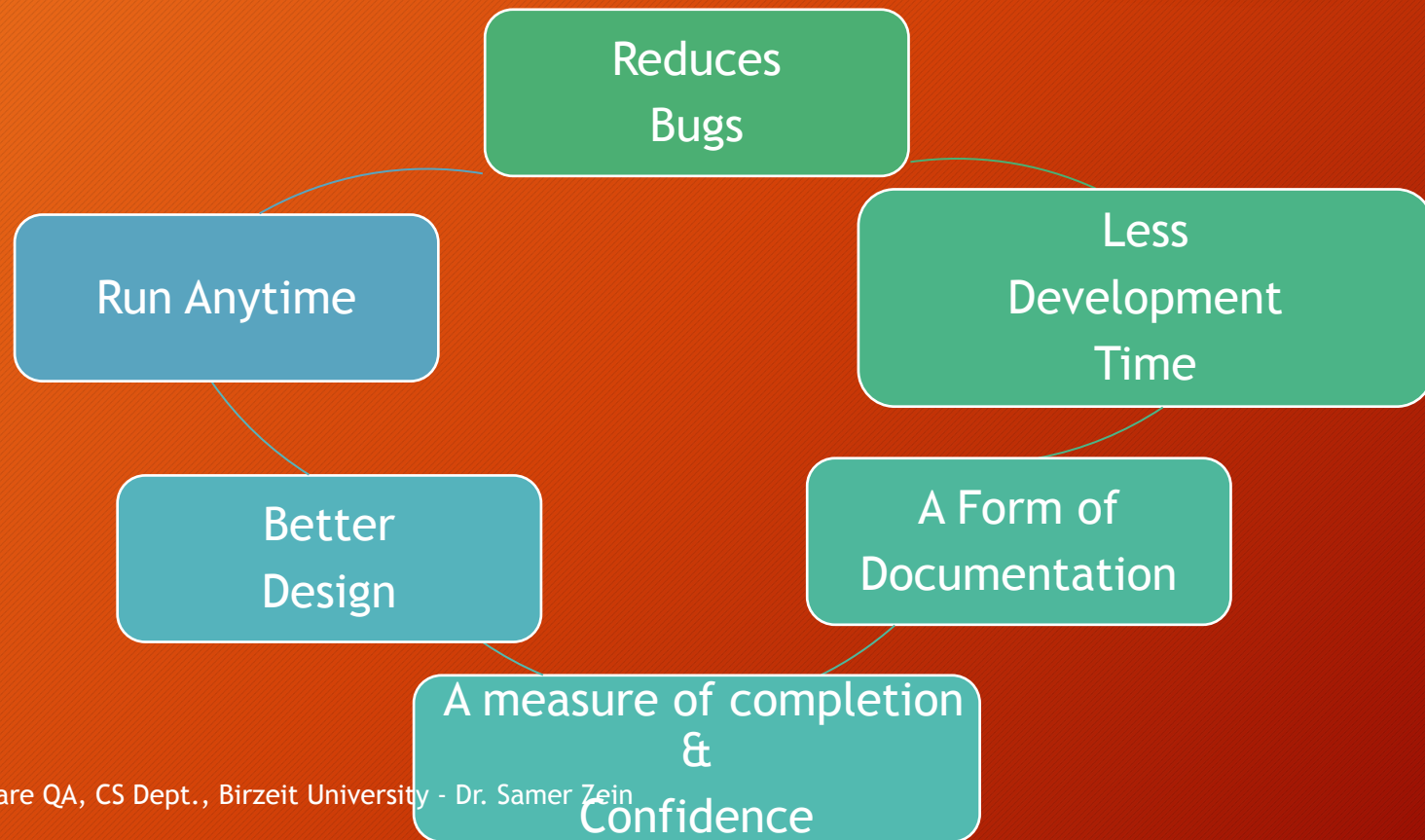
- *A unit test is a piece of code written by a developer that exercises a very small, specific area of functionality of the code being tested.*
- Usually a unit test exercises some particular method in a particular context.
- For example, you might add a large value to a sorted list, then confirm that this value appears at the end of the list.
- Or you might delete a pattern of characters from a string and then confirm that they are gone.

Software QA, CS Dept., Birzeit University - Dr. Samer Zein

Unit Testing..2

- All what we need here is to prove/ have some confidence, that our code does what we intend!
 - *and so we want to test very small, very isolated pieces of functionality!*
- By building up confidence that the individual pieces work as expected, we can then proceed to assemble and test working systems.
 - *After all, if we aren't sure the code is doing what we think, then any other forms of testing may just be a waste of time.*

Why Unit Testing?



Software QA, CS Dept., Birzeit University - Dr. Samer Zein

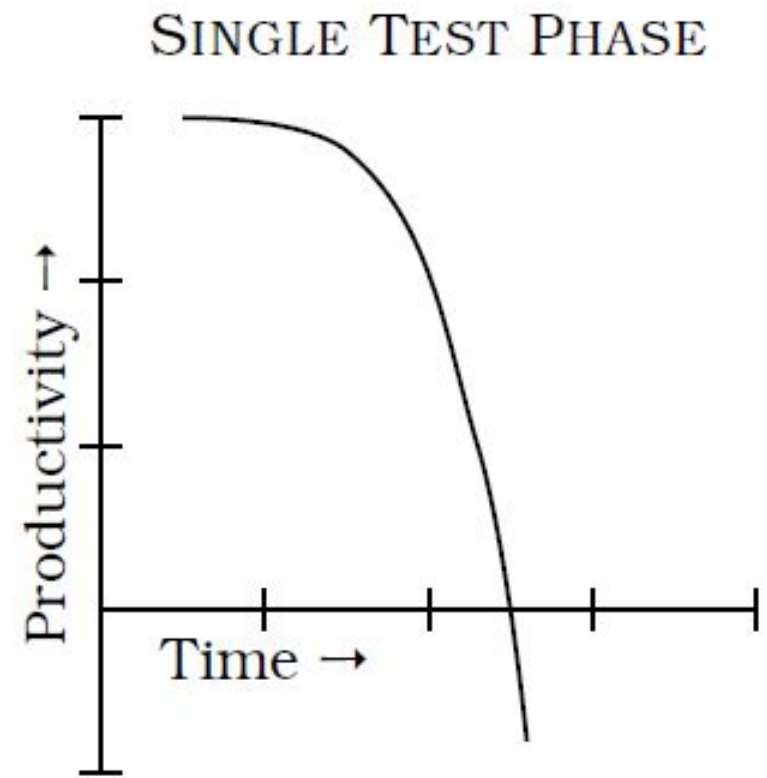
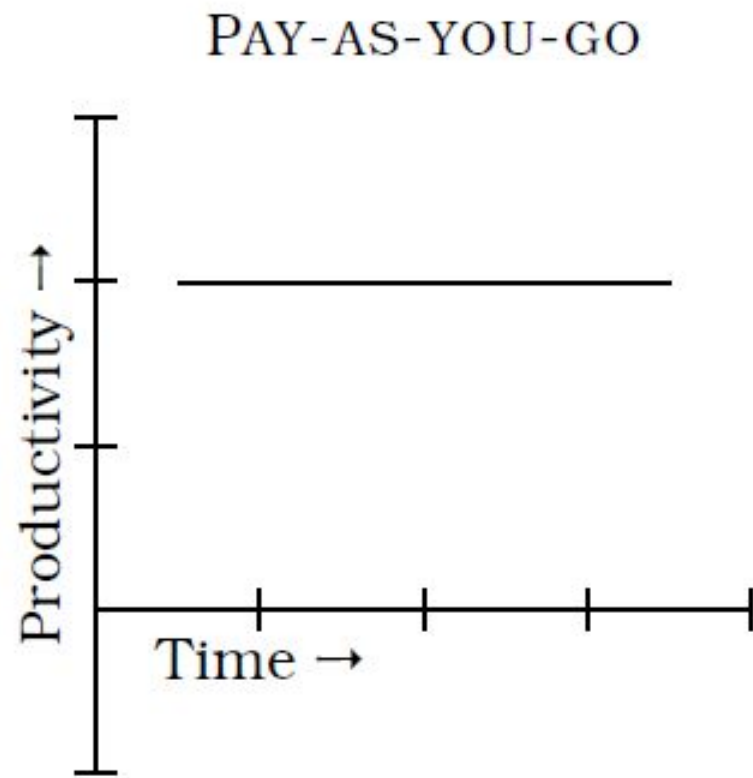
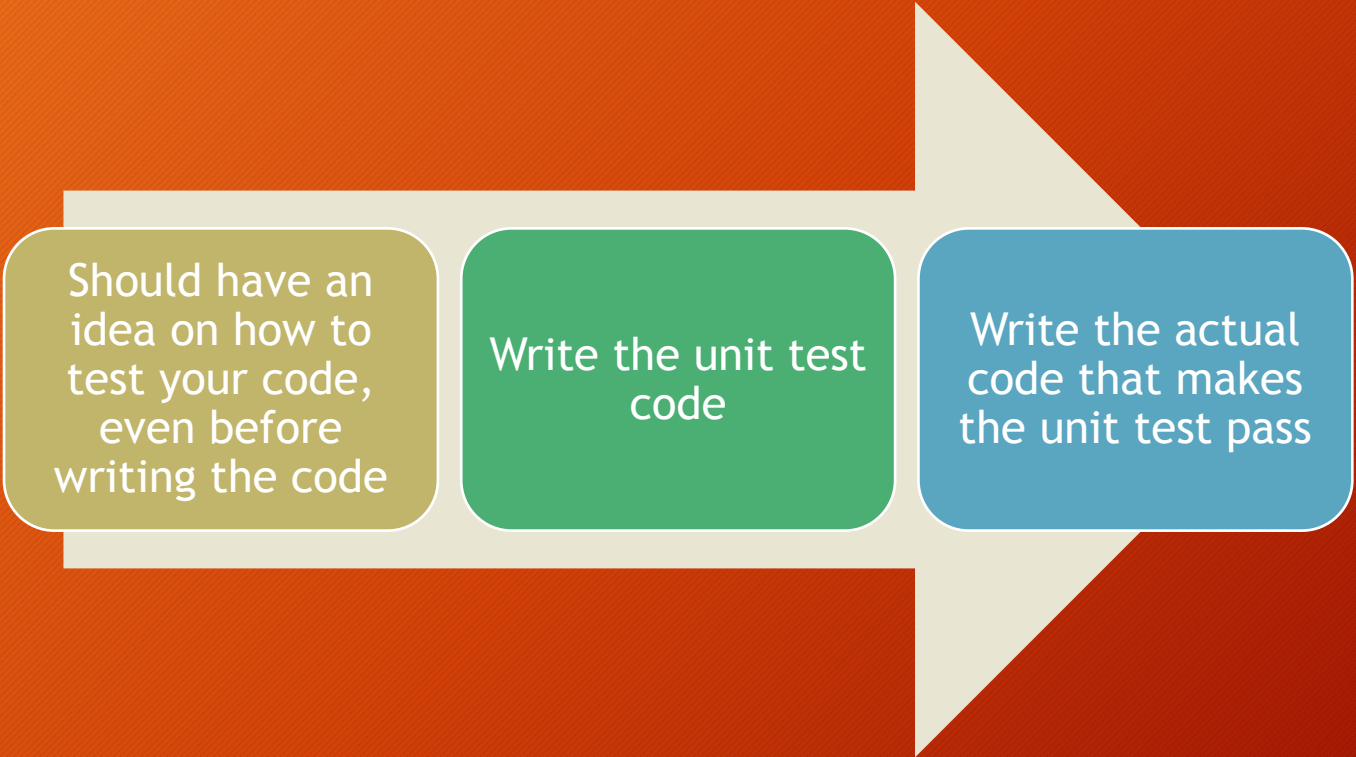


Figure 1.1: Comparison of Paying-as-you-go vs. Having a Single Testing Phase

Unit Testing General Process



Should have an idea on how to test your code, even before writing the code

Write the unit test code

Write the actual code that makes the unit test pass

Unit Testing: the How!

- To check the code if it is behaving as expected, we use an **assertion**, a simple method call that verifies that something is true.

```
public void assertTrue(boolean condition) {  
    if (!condition) {  
        abort();  
    }  
}
```


Let's Have an Example

```
Line 1  public class Largest {  
-  
-      /**  
-      * Return the largest element in a list.  
5      *  
-      * @param list A list of integers  
-      * @return The largest number in the given list  
-      */  
-      public static int largest(int[] list) {  
10         int index, max=Integer.MAX_VALUE;  
-         for (index = 0; index < list.length-1; index++) {  
-             if (list[index] > max) {  
-                 max = list[index];  
-             }  
15         }  
-         return max;  
-     }  
- }
```

• [7, 8, 9] → 9

• [8, 9, 7] → 9

• [9, 7, 8] → 9

• [7, 9, 8, 9] → 9

• [1] → 1

• [-9, -8, -7] → -7

Activity: What are the test cases you would think about?

Software QA, CS Dept., Birzeit University - Dr. Samer Zem


First Unit Test Example:

```
import junit.framework.*;
public class TestLargest extends TestCase {
    public TestLargest(String name) {
        super(name);
    }
    public void testSimple() {
        assertEquals(9, Largest.largest(new int[] {7,9,8}));
    }
}
```



There was 1 failure:
1) testSimple(TestLargest)junit.framework.AssertionFailedError:
expected:<9> but was:<2147483647>
at TestLargest.testSimple(TestLargest.java:10)

Zooming on Error

```
Line 1  public class Largest {  
-  
-      /**  
-      * Return the largest element in a list.  
5      *  
-      * @param list A list of integers  
-      * @return The largest number in the given list  
-      */  
-      public static int largest(int[] list) {  
10      int index, max=Integer.MAX_VALUE;   
-      for (index = 0; index < list.length-1; index++) {  
-          if (list[index] > max) {  
-              max = list[index];  
-          }  
15      }  
-      return max;  
-      }  
-  }  
Softw
```

Fix, and continue testing

```
public void testOrder() {  
    assertEquals(9, Largest.largest(new int[] {9,8,7}));  
    assertEquals(9, Largest.largest(new int[] {8,9,7}));  
    assertEquals(9, Largest.largest(new int[] {7,8,9}));  
}
```




```
There was 1 failure:  
1) testOrder(TestLargest)junit.framework.AssertionFailedError:  
    expected:<9> but was:<8>  
    at TestLargest.testOrder(TestLargest.java:4)
```

Software QA

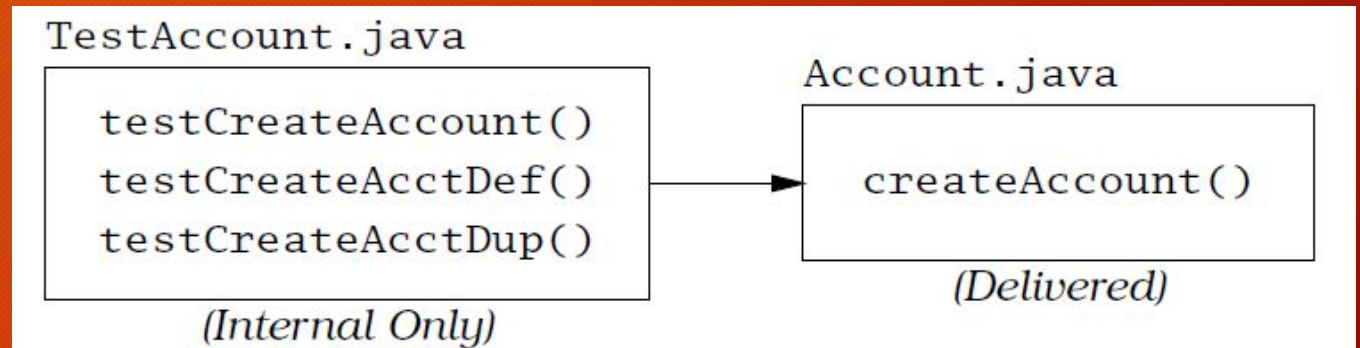
Zooming on Error

```
Line 1  public class Largest {  
-  
-      /**  
-      * Return the largest element in a list.  
5      *  
-      * @param list A list of integers  
-      * @return The largest number in the given list  
-      */  
-      public static int largest(int[] list) {  
10      int index, max=Integer.MAX_VALUE;  
-      for (index = 0; index < list.length-1; index++) {  
-          if (list[index] > max) {  
-              max = list[index];  
-          }  
15      }  
-      return max;  
-      }  
-  }
```



Naming Conventions

- If you have a method named “createAccount()” that you would like to test, then the test method should be
 - testCreateAccount()
- Of course you can have as many test methods as you can for the same tested method.
- The unit tests are internal only, i.e, never shipped to customer



In General, the Test Code Should:

- Set up all conditions needed for testing (create any required objects, allocate any needed resources, etc.)
- Call the method to be tested
- Verify that the method to be tested functioned as expected
- Clean up after itself

JUnit Assertions

- They let you assert that some condition is true; that two bits of data are equal, or not, and so on.
- All the following Assert methods will
 - report failures when the assertion is false.
 - Report error when an exception is thrown

Assertions..1

assertEquals

```
assertEquals([String message],  
             expected,  
             actual)
```

Optional

```
assertEquals([String message],  
             expected,  
             actual,  
             tolerance)
```

Precision
For floating
points

Assertions..2

assertNull

```
assertNull([String message], java.lang.Object object)  
assertNotNull([String message], java.lang.Object object)
```

Asserts that the given object is null (or not null), failing otherwise. The message is optional.

assertSame

```
assertSame([String message], expected, actual)
```

Asserts that *expected* and *actual* refer to the same object, and fails the test if they do not. The message is optional.

```
assertNotSame([String message], expected, actual)
```


Assertions..3

- Normally you will have several assertions at one test method. If one assertion fails, the rest are aborted.
- However, the rest of test methods will continue to work.

assertTrue

```
assertTrue([String message], boolean condition)
```

In addition to testing for true, you can also test for false:

```
assertFalse([String message], boolean condition)
```

fail

```
fail([String message])
```

Fails the test immediately, with the optional message. Often used to mark sections of code that should not be reached (for instance, after an exception is expected).

Per-Test Setup and Tear-Down

- Important Note: *Each test should run independently of every other test; this allows you to run any individual test at any time, in any order.*
- To accomplish this feat, you may need to reset some parts of the testing environment in between tests, or clean up after a test has run

Per-Test Setup and Tear-Down..2

- Remember, every unit test framework (JUnit, NUnit, Visual Studio, etc) has its own methods.
- But they all serve the same purpose, thus you should refer to their documentation.
- In JUnit,
 - **protected void setUp()**
 - Called before each test method
 - **protected void tearDown()**
 - Called after each test method

Per-Test Setup and Tear-Down Example

```
public class TestDB extends TestCase {
    private Connection dbConn;

    protected void setUp() {
        dbConn = new Connection("oracle", 1521,
                                "fred", "foobar");
        dbConn.connect();
    }

    protected void tearDown() {
        dbConn.disconnect();
        dbConn = null;
    }

    public void testAccountAccess() {
        // Uses dbConn
        XXX XXXXXX XXX XXXXXX,
        XXX XXX XXXX XXXX XXX XXXX;
    }

    public void testEmployeeAccess() {
        // Uses dbConn
        XXX XXX XXXXXX XXX XXXXXX,
        XXXX XXX XXX XXXX XXX XXXX,
    }
}
```

In this example, `setUp()` will be called before `testAccountAccess()`, then `tearDown()` will be called. `setUp()` will be called again, followed by `testEmployeeAccess()` and then `tearDown()` again.