UNIVERSITY of BRADFORD

Software Testing

Testing

- Testing is the key method for verifying the correctness of a program or software system
- Classical project development methodologies, especially waterfall approach, use testing at the end of the development stage, sometimes at the very end of the project
- Agile methods recommend the use of testing in a very rigorous way and as early as possible
- According to XP methodology, test set construction takes precedence over programming (*test first programming* and test-driven development)

Testing

- Testing is a destructive activity, not always popular, but very important and necessary
- Exhaustive testing (testing all possible values and program/system behaviour) is intractable
- Need certain principles: code coverage, branch coverage, partition the set of inputs into equivalence classes etc.
- Code coverage means providing inputs such that every instruction is executed at least once
- Any testing method aims at building a complete test set (set of input data fulfilling all the criteria of a testing principle)

EnterprisePro Testing Methods

- During the implementation of each unit of code, either
  - Unit testing using code coverage
  - Systematic code inspection
- Before releasing the code
  - Acceptance testing (remember requirements document)
- Deliverables:
  - Unit testing code
  - Code inspection
  - Acceptance testing
  - Testing document

Unit Testing

Unit Testing

- This is a white-box testing strategy, addressing the smallest programming units
- It is conceived by each programmer – sometimes before programming starts (XP programming advocates 'testing-first programming': test set built before programming starts !)
- Object oriented style requires testing at the class level: each method of a class is tested
- It does not follow any specific testing strategy – this depends on each programmer
- Languages supporting Unit Testing: Java, C, Python, Ruby, PHP…

Java Unit Testing

- Called JUnit
- Written in java
- Website: junit.org
- Use it systematically for each method of your code such that code coverage is achieved – Software Design and Development

JUnit – Basic Principle

- Each method of a class is tested by writing a test method checking that some expected values are obtained or not.
- It uses an assert statement and the annotation @Test that checks the validity of the results
- JUnit testing means writing appropriate code executing and testing java methods and classes

JUnit Code

```
public class MyClass {                          // class to be tested
    public int multiply (int x, int y) {        // method to be tested
        return x * y;
    }
}
package myClassWithUnitTest;            //recommended code structure
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class MyClassTest {                      // Junit code associated with MyClass
    @Test
    public void testMultiply() {                // testing multiply (int x, int y)
        MyClass tester = new MyClass();
        assertEquals("10*5 = 50", 50, tester.multiply(10, 5));
    }
}
```

Testing with JUnit

- JUnit creates a new instance of the test class before invoking each @Test method. To perform test validation, we use the assert methods provided by the JUnit Assert class (`public class Assert extends java.lang.Object`). Some of the most popular assert methods are:
- **assertEquals("message", A, B)**: Asserts the equality of objects A and B. This assert invokes the equals() method on the first object against the second.
- **assertArrayEquals("message", A, B)**: Asserts the equality of the A and B arrays.
- **assertTrue("message", A)**: Asserts that the A condition is true.
- **assertFalse("message", A)**: Asserts that the A condition is false.
- **assertNull("message", A)**: Asserts that the object A is null.
- **assertNotNull("message", A)**: Asserts that the A object is not null.

More on JUnit

- JUnit tutorials at:
  http://www.vogella.com/tutorials/JUnit/article.html

- Use 1st year Java programming lecture notes (SDD)

- JUnit testing prototypes automatically generated (require proper assertions)

- Other JUnit examples:
  - See Additional Examples in Canvas (classes Account & Complex – handling complex numbers); it also shows how to use automatically generated test classes

PHPUnit – Basic Principles

- Similar to JUnit:
  - Each method of a class is tested by writing a test method checking that some expected values are obtained or not.
  - It uses an assert statement that checks the validity of the results

- **Note.** PHPUnit requires the use of objects (remember design for test condition) – very important!

PHPUnit - Example

- A class Calculator for simple arithmetic operations – only `add` illustrated

```
<?php
class Calculator{                                   // PHP class
  public function add($numbers_to_add){             // method/function to be tested
      $sum = 0;
      foreach($numbers_to_add as $num){
         $sum = $num + $sum;
      }
      return $sum;
  }
public function multiply($numbers_to_multiply){ // method/function to be tested
      $product = 0;
      foreach($numbers_to_multiply as $num) {
         $product = $num * $product;
      }
      return $product;
  }
?>
```

PHPUnit - Testing

- Test class for Calculator, called `CalculatorTest` with function `testAdd` and `testMultiply`.

```
<?php
require_once('calculator.php');                          //the calculator class
class CalculatorTest extends PHPUnit_Framework_TestCase{  // testing class
public function testAdd(){                                // testing function
  $calc = new Calculator();
  $sum = $calc->add(array(2,0,1,6));
  $this->assertEquals(9, $sum);    // use assert to check if 2+0+1+6 is equal to 9
}
public function testMultiply(){
$calc = new Calculator();
$product = $calc->multiply(array(1,5,4,3));
$this->assertEquals(60, $product);   //check if 1*5*4*3 is equal to 60
}
?>
```

PHP Unit with Functions

- When PHP is not OO then PHP unit testing with functions:
  - create functions out of the PHP code
  - build assert functions
- Both user functions and assert functions have to be built !!
- More on Unit Testing with functions; example using square() & palindrome functions:

  PHPUnitTesting-Functions.php

PHP Function & assert- Example

```
<?php
…
function square($number) {          // PHP function
   return ($number * $number);
}
…
?>
```

- An assertEquals like function is built

```
function assertEquals($input, $expectedOutput) {   // PHP assertEquals
     if($input == $expectedOutput) {
          echo 'SUCCESS';  return TRUE; }
     else {
          echo 'FAILED'; return FALSE; }
}
```

PHP Function & assert- Example

```
<?php
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php
        echo '<h1>Square Test</h1>';
        echo assertEquals(4, square(2));
        ?>
    </body>
</html>
```

More on PHPUnit

- PHPUnit: https://phpunit.de/manual/current/en/phpunit-book.pdf

  -- more advanced features

Code Inspection

Code Inspection

- For somecode, such as set of static webpages, HTML, non-OO code, use
  - Code inspection
- Old approach on static verification: checks systematically the code against some criteria. Predecessor of unit testing
- According to some surveys: Code inspection discovers on average 65% of coding faults; and identifies up to 20% of requirements faults and 40% of design faults.
- How is it applied to static webpages?

Code Inspection Principles

- Two key aspects are ‘inspected’ on each webpage:
  - Webpage layout and structure
  - Links to other webpages (correct)
- Layout
  - consistent use of format, size, colours, menus
  - consistent use of set of images and text, logo etc
- Structure
  - type of structure: landing page, presentation, menu

Code Inspection

- checks systematically the code against some criteria –
  - algorithmic correctness (e.g., receiving valid input, terminates, returns the correct output,);
  - layout;
  - coding standards;
  - exit conditions for while-loops or class methods/operations
  - etc.

Acceptance Testing

Acceptance Testing

- This is aimed at testing that the software systems and its components work as expected
- The test set is built in accordance with a **requirements specification document**
- Test set is built starting from the set of functional requirements:
  - Each functional feature has a set of test sequences
  - The test set might be built incrementally according to the project management approach
- Other test sequences might be built (non-functional features)
- Use the Library System for UG/PG dissertations requirements document (see Canvas Week 3-4.)

Simple Example

- Functions and their description
- **1. Login/logout for LS administrator and assistants**. LS will have an administrator who will login into the system with *a user name and a passwor*d. Each LS assistant will also login with similar credentials. These are validated and if not correct they have to be re-entered. Administrator/Assistant exits the system by logging out.
- Generate test sequences in a table, function by function
- For the Login function the test set consists of test sequences for
  - Correct inputs: *name & password*
  - Three combinations of wrong inputs for *name, password*

Acceptance Testing Document

- This stage should produce a document showing the results of the acceptance testing with reference to requirements

Input sequence	Expected result	Current output	Comments
Req 1: Login			
Correct login: valid_name, valid_psswd	Login accepted	…	Passed
Incorrect login: valid_name, wrong_psswd	Login incorrect		Passed
…			
Req2: …			

- **Note.** Input sequences must consists of input values, i.e., valid_name = “Adam XYZ”; password =“ac12£%??”

Building the Test Set

- Each function from the Requirements document is analysed and input data identified
- Correct and incorrect values are considered
- Consider the *upload* function

Building the Test Set - Example

- **upload**: One of the LS assistants is uploading for each dissertation the following information: *dissertation author; student University ID; year when the dissertation has been submitted; dissertation title; dissertation ID; dissertation types; dissertation* – a PDF copy of the dissertation.
- Test sequences made out of a set of
  - Correct input values (*dissertation author* = "Adam XYZ"; *UB number* = 210028567; year = 2021; …)
  - Incorrect input data
    - empty values (no value for name or dissertation title…)
    - *year* or *student ID* is not numeric
    - *student ID* does not exist
    - …

Conclusions

- Unit testing; JUnit, PHPUnit
- Code inspection for static webpages & non-OO code
- Acceptance testing – requirements document
- Deliverables
  - Unit testing: set of unit testing classes
  - Code inspection
  - Acceptance testing