Software Testing

CS 102 – Algorithms and Programming II
Spring 2006
Software Development Lifecycle

General Waterfall Model

- Requirements
- Design
- Implementation
- Testing
- Maintenance
Testing

- revealing defects
- evaluating quality
- related to verification and validation
  - Verification: does the software satisfy the conditions imposed at the beginning?
  - Validation: does the software satisfy specified requirements?
Related Terms

- **Test Case**
  - A set of inputs
  - Execution conditions
  - Expected outputs
- **A test suite** is a set of test cases, or a set of test cases and test procedures.
Related Terms

- **Error**
  - a mistake, misconception, or misunderstanding on the part of a software developer

- **Fault (defect)**
  - is an anomaly in the software that may cause it to behave incorrectly, and not according to its specification

- **Failure**
  - inability of a software system or component to perform its required functions within specified performance requirements
Types of Faults

- Algorithmic faults
- Computation and precision faults
- Documentation faults
- Stress or overload faults
- Capacity and boundary faults
- Timing or coordination faults
- Throughput or performance faults
- Recovery faults
- …
Software Testing Principles

- Testing is a creative and challenging task
- Testing must be carried out by a group independent of the development group
- The probability of existence of additional defects is proportional to the number of already detected defects
- Testing should not be done with the assumption that no errors will be found
Software Testing Principles

- A good test case has a high probability of revealing a yet-undetected defect
- A test case must contain expected results if possible
- Test cases should be developed for both valid and invalid input conditions
- The software should be static during testing
Testing Life Cycle

- Establish test objectives
- Design test cases
- Write test cases
- Test test cases
- Execute tests
- Evaluate test results
Types of Testing

- Unit testing
- Integration testing
- Function testing
- Performance testing
- Acceptance testing
- Installation testing

A functioning system

A verified system

A validated system
How to Design Test Cases?

- Test all possible inputs or user actions!

infeasible
and
unnecessary
How to Design Test Cases?

- Black-box testing

Inputs → ? → Outputs
How to Design Test Cases?

- White-box testing

```
POINTER = FALSE

X > K

X = X - 1

CALL SUB(X, POINTER, RESULT)

RESULT > 0

POINTER = TRUE

NO

YES

PRINT RESULT
```

as taken from Pfleeger, 2002
Simple Input Choice Example

- A component is coded to expect a positive as the input value
  - A very large positive integer within the range of input (i.e. 32767 if 16 bit signed int range)
  - A positive integer
  - A positive fixed-point decimal
  - A number greater than 0 but less than 1
  - Zero
  - A negative number
Unit Testing in Java

- **JUnit**: a simple unit testing framework for Java

- **Contains:**
  - Assertions for testing expected results
  - Test fixtures for sharing common test data
  - Test runners for running tests

http://www.junit.org
JUnit

Welcome to JUnit.org. This site is dedicated to software developers using JUnit or one of the other XUnit testing frameworks. We'll be adding more content and web-based services over time. Initially we’ll be providing links to give you a one-stop destination to learn the latest information on unit testing.

Our goal is to serve you. Please tell us what you'd like to see here by contacting us at junit@objectmentor.com. JUnit support is handled through the JUnit Yahoo Group.

If you are looking for another one of the testing frameworks, you should look on www.xprogramming.com under software.

Overview

JUnit is a regression testing framework written by Erich Gamma and Kent Beck. It is used by the developer who implements unit tests in Java. JUnit is Open Source Software, released under the Common Public License Version 1.0 and hosted on SourceForge.
JUnit

JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.

- Getting Started
- Documentation
- JUnit related sites/projects
- Mailing Lists
- Get Involved

Getting Started

To get started with unit testing and JUnit read the article: JUnit Cookbook. This article describes basic test writing using JUnit 4.

You find additional samples in the org.junit.samples package:

- SimpleTest.java - some simple test cases
- VectorTest.java - test cases for java.util.Vector

JUnit 4.0 only comes with a textual TestRunner. For graphical feedback, most major IDE's will support JUnit 4 soon. In the meantime, you can run JUnit 4 tests in a JUnit 3 environment by adding the following method to each test class:

```java
public static Test suite() {
    return new JUnit4TestAdapter(ThisClass.class);
}
```
How to Install JUnit

- To install JUnit on Windows
  - Unzip the junit.zip distribution file to a directory referred to as `%JUNIT_HOME%`.
  - Add JUnit to the classpath:
    ```
    Set CLASSPATH=%CLASSPATH%;%JUNIT_HOME%\junit.jar
    ```
public void testGet_upperBoundary() {
    System.out.println("\ttestGet: upper boundary case\n" +
    int index = testBag.valid - 1;
    int nINPUT: " + index + 
    OR FAILURE: the returned n
    The value of elemen" +
    testBag.get(index) ==

    testGet: ordinary case":
    OR FAILURE: the returned n
    The value of elemen" +
    testBag.get(index) ==

    JUnit4TestAdapter(TestIntE

    public static void main(String args[])
    org.junit.runner.JUnitCore.main("TestIntBagFeatures");
}
Example: Test Suite for IntBag class

- create a class which will hold the test cases:

```java
import org.junit.*;
import static org.junit.Assert.*;
import java.util.*;

public class TestIntBagFeatures {
    
}
```
Example: Test Suite for IntBag class

- create a test fixture if you have tests for a set of common objects

```java
@Before public void setUp() {
    testBag = new IntBag();

    // initialize the IntBag
    ...
}
```
Example: Test Suite for IntBag class

- create a test fixture if you have tests for a set of common objects

```java
@After public void tearDown() {
    testBag = null;
}
```
Example: Test Suite for IntBag class

- The test methods are annotated with `@Test`

```java
@Test
public void testSomething() {
    ...
}
```
Example: Test Suite for IntBag class

- **Test case: Construction**
  - **Input: no parameters**
  - **Execution condition: N/A**
  - **Expected result: successfully initialize an IntBag**

```java
@Test
public void testEmptyBag() {
    IntBag testBag = new IntBag();
    assertEquals("Test failed...", 0, testBag.size());
}
```
Example: Test Suite for IntBag class

- `assertFalse(java.lang.String message, boolean condition)`
- `assertNotNull(java.lang.String message, java.lang.Object object)`
- `assertSame(java.lang.String message, java.lang.Object expected, java.lang.Object actual)`
- `assertTrue(java.lang.String message, boolean condition)`
Example: Test Suite for IntBag class

- Test case: Construction
  - Input: -1 as the size of IntBag (illegal)
  - Execution condition: N/A
  - Expected result: The program should throw a negative array size exception

```java
@Test(expected = NegativeArraySizeException.class)
public void testUnderflowConstruction() {
    IntBag testBag = new IntBag(-1);
}
```
Example: Test Suite for IntBag class

- Test case 3: Remove from index
  - Input: 3 as the index to be removed (illegal)
  - Execution condition: the IntBag is empty
  - Expected result: nothing happens
Example: Test Suite for IntBag class

- Test case 4: Remove from index
  - Input: -1 as the index to be removed (illegal)
  - Execution condition: the IntBag has 5 elements
  - Expected result: nothing happens
Example: Test Suite for IntBag class

- Test case 5: Remove from index
  - Input: 10 as the index to be removed (illegal)
  - Execution condition: the IntBag has 5 elements
  - Expected result: nothing happens
Example: Test Suite for IntBag class

Test case 6: Remove from index

- Input: 0 as the index to be removed (boundary input)
- Execution condition: the IntBag has 5 elements
- Expected result: the number at index 0 is removed, the valid value is decremented, and the rest of the numbers remain in the bag.
Example: Test Suite for IntBag class

- Test case 7: Remove from index
  - Input: 4 as the index to be removed (boundary input)
  - Execution condition: the IntBag has 5 elements (valid value=4)
  - Expected result: the number at index 4 is removed, the valid value is decremented, and the rest of the numbers remain in the bag.
Example: Test Suite for IntBag class

- Test case 8: Remove from index
  - Input: 1 as the index to be removed (ordinary input)
  - Execution condition: the IntBag has 5 elements (valid value=4)
  - Expected result: the number at index 1 is removed, the valid value is decremented, and the rest of the numbers remain in the bag.
Example: Test Suite for IntBag class

JUnit version 4.0

OK (24 tests)

Press any key to continue...
assertFalse("Test Contains failed:
CONDITION: " +
" the bag is empty " +
"\nREASON FOR FAILURE: although the " +
"bag is empty " +
"the method returns true.",
testBag.contains(numberToBeSearched));
JUnit version 4.0

************E******

Time: 0

There was 1 failure:
1) testContains(TestIntBagFeatures)
   java.lang.AssertionError: Test Contains failed:
   CONDITION: the bag is empty
   REASON FOR FAILURE: although the bag is empty the method returns true.
   at org.junit.Assert.fail(Assert.java:58)
   at org.junit.Assert.assertTrue(Assert.java:27)
   at TestIntBagFeatures.testContains(TestIntBagFeatures.java:642)
   at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
   at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
   at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
   at java.lang.reflect.Method.invoke(Method.java:585)
   at org.junit.internal.runners.TestMethodRunner.executeMethodBody(TestMethodRunner.java:99)
   at org.junit.internal.runners.TestMethodRunner.runUnprotected(TestMethodRunner.java:81)
   at org.junit.internal.runners.BeforeAndAfterRunner.runProtected(BeforeAndAfterRunner.java:34)
   at org.junit.internal.runners.TestMethodRunner.runMethod(TestMethodRunner.java:45)
   at org.junit.internal.runners.TestClassMethodsRunner.invokeTestMethod(TestClassMethodsRunner.java:71)
   at org.junit.internal.runners.TestClassMethodsRunner.run(TestClassMethodsRunner.java:35)
   at org.junit.internal.runners.TestClassRunner$1.runUnprotected(TestClassRunner.java:42)
   at org.junit.internal.runners.BeforeAndAfterRunner.runProtected(BeforeAndAfterRunner.java:34)
   at org.junit.internal.runners.TestClassRunner.run(TestClassRunner.java:52)
   at org.junit.internal.runners.CompositeRunner.run(CompositeRunner.java:29)
When to Stop?

- All planned tests have been executed and passed
- All specified coverage / functionality / requirements goals have been met
- Rate of fault detection is below a threshold
- Deadlines
- Test budget used up
References


References

- http://junit.sourceforge.net/
- http://www.junit.org
Online Testing Resources

- http://www.softwareqatest.com/qatfaq2.html#FAQ2_6
- http://www.testingeducation.org/articles/domain_testing_cseet.rtf
- http://www.testingeducation.org/articles/scenario_intro_ver4.doc
- http://www.testingeducation.org/coursenotes/kaner_cem/ac_200108_blackboxtesting/