

Testing—Part 2

Lecture #14
Software Engineering and
Project Management

Instructed by Steven Choy on Feb 5, 2007



1

System Testing



2

System Testing



- System testing ensures the system, as a whole, complies with both functional and non-functional requirements
- Activities involves:
 - Functional Testing
 - Performance Testing
 - Pilot Testing
 - Acceptance Testing
 - Installation Testing

3

Functional Testing



- Also known as Requirement Testing
- As the name reveals, it checks if the system conforms with the functional requirement
- Test cases are identified from use case model and requirement specification

4

Performance Testing



- What are our goals?
 - Check if our system performance meets user's requirement
 - Find out the details of the system in terms of the following metrics:
 - Throughput (no. of request/sec)
 - CPU/Memory/IO/Network Utilization
 - Maximum number of client/request it can handle before breaking the system
 - Time for fail-over of systems
 - Security level (How vulnerable is our system for attack?)
 - etc...

5

Performance Testing



- Load Testing
 - Load the system with an expected number of data/request and check if the system can handle
- Stress Testing
 - Push the system beyond the limit to evaluate its robustness
 - Load the system with an unreasonable large number of request/data, trying to stress the system to the breaking point
- Timing Testing
 - Attempt to find behaviours that violate timing constraints described by the non-functional requirement
- Security Testing
 - Try to find out the security flaws of the system
- Recovery Testing
 - Evaluates the ability of the system to recover from erroneous state (e.g. hardware/network failure)

6

Performance Test Tools



- ApacheBench from Apache HTTP server
- Rational Performance Tester from IBM
- LoadRunner from Mercury
- QALoad from Compuware

7

Acceptance Testing



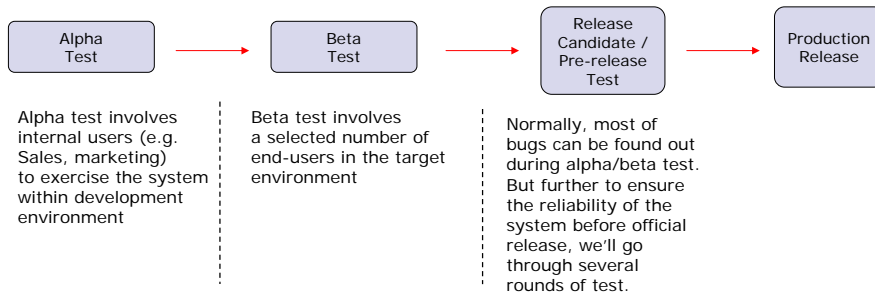
- "Acceptance" – does your **client** accept the system? → **Acceptance Testing**
 - Acceptance testing is performed by clients, not developers
 - The client designs and performs series of tests to check if it conforms with the requirement
- Typically, majority of bugs are found during this stage (**WHY?**)
- Besides, after trying out the system, clients may request for change of requirement in this stage

8

Pilot Testing



- Pilot testing is “Let end-users really try out the system before its official release”



9

Does Testing end Here?



10

Regression Testing



- Object-oriented software development is an iterative process
- Fix one bug may introduce another bug somewhere in the code
- Adding/modifying a feature may also introduce side effect to the existing code
- A system that fails after the modification of a component is said to be **regress**
- Tests that try to produce such failure → **Regression Test**

11

Regression testing...



- Regression testing is testing that a program has not regressed; the functionality that was working yesterday is still working today. In other word, regression testing is to ensure that we do not introduce new bugs while removing existing bugs. A program still works correctly after changes were made to it.

12

Regression Testing



- So, what's the best way to ensure the modified component does not introduce side effect to the existing system?
 - **Retest the whole system with all the test cases!**
- Strategies to streamline the process:
 - Retest dependent components
 - Retest risky components
 - Retest frequently-used components

13

"Yes, I admit Regression Testing is crucial to our system"

But...

How can we perform all test cases again & again **manually**?

It's really time-consuming & a waste of development man-hours

14

Automate Your Testing



- Unit/Integration Test Automation
 - JUnit – an open-sourced test framework for Java
 - Latest version is 4.0 but we only focus on v3.8
- User Interface Testing Automation
 - QARun from Compuware
 - QuickTest from Mercury
 - Rational Robot from IBM
 - AutomatedQA

15

A Very Simple Example



```
public class Calculator {
    public Calculator() {
    }

    public int add(int num1, int num2) {
        return num1 + num2;
    }

    public int minus(int num1, int num2) {
        return num1 - num2;
    }
}
```

16

Unit Test in a manual way



- If you need to perform unit testing on that piece of code, you probably do it using "System.out"...

```
public class ManualTestCalculator {
    public static void main(String[] args) {
        Calculator calculator = new Calculator();
        int result = calculator.add(100, 200);
        System.out.println("Result for (100+200) = " + result);
        result = calculator.minus(200, 100);
        System.out.println("Result for (200-100) = " + result);
    }
}
```

Console Output

```
Result for (100+200) = 300
Result for (200-100) = 100
```

17

Automate Unit Testing by JUnit



- JUnit is developed to automate unit testing in Java
- A regression testing framework, written by Erich Gamma and Kent Beck

18

JUnit Example



```
import junit.framework.*;

public class CalculatorTest extends TestCase {
    private Calculator calculator = null;

    public CalculatorTest(String name) {
        super(name);
    }

    protected void setUp() {
        calculator = new Calculator();
    }

    public static Test suite() {
        TestSuite suite = new TestSuite(CalculatorTest.class);
        return suite;
    }
}
```

19

JUnit Example (Con't)



```
/** Test of add method, of class Calculator */
public void testAdd() {
    /* Test case #1: Adding two positive integers together */
    int result = calculator.add(100, 200);
    assertEquals(300, result);

    /* Test case #2: Adding two negative integers together */
    result = calculator.add(-100, -200);
    assertEquals(-300, result);
}

/** Test of minus method, of class Calculator */
public void testMinus() {
    /* Test case #1: Subtract two positive integers */
    int result = calculator.minus(200, 100);
    assertEquals(100, result);

    /* Test case #2: Subtract two negative integer */
    result = calculator.minus(-200, -100);
    assertEquals(-100, result);
}
}
```

20

Unit Testing Quotes



- Unit tests can be tedious to write, but they save you time in the future by catching bugs after changes [William Wake]

21

JUnit References



- Download JUnit at:
<http://www.junit.org>
- Learn more about JUnit at:
<http://junit.sourceforge.net/doc/testinfected/testing.htm>

22

Documenting Your Test



23

Test Documentation



- Test Plan
 - Describe **what** to test
 - A document describes what to be tested, the scope, approach, resources and schedule of the test
- Test Specification
 - Describe **how** to test
 - Describe how each test case is run
- Test Report
 - Present the test result

24

Sample Test Plan Outline



BACKGROUND	TESTING TASKS Functional tasks (e.g., equipment set up) Administrative tasks
INTRODUCTION	RESPONSIBILITIES Who does the tasks in Section 10? What does the user do?
ASSUMPTIONS	SCHEDULE
TEST ITEMS List each of the items (programs) to be tested.	RESOURCES
FEATURES TO BE TESTED List each of the features (functions or requirements) which will be tested or demonstrated by the test.	APPROVALS
FEATURES NOT TO BE TESTED Explicitly lists each feature, function, or requirement which won't be tested and why not.	
APPROACH Describe the data flows and test philosophy. Simulation or Live execution, Etc.	
ITEM PASS/FAIL CRITERIA Blanket statement Itemized list of expected output and tolerances	
TEST DELIVERABLES What, besides software, will be delivered? Test report Test software	

25

Sample Test Specification Outline



Test case ID
Purpose What are the test items? Basically what items are you testing? If in same document as above, then a simple 2 line overview and reference to previous document is sufficient for our purpose.
Source of the feature Who originated the this test case?
Programmer ID Who programmed the code implementing the feature?
Environmental Setup Any special procedures required for the individual tests?
Test design method What techniques has been used? What coverage criterion will be met? Did you use [J/Cpp]Unit? How do you organize the test cases using [J/Cpp]Unit? Many test cases differ only in a few parameters (.e. shelf/slot). These should be parameterized with [J/Cpp]Unit and invoked as needed. Also, there would be some type of system startup (initialize system to a known state and then a tear down part - restore system to initial state after test), that should be found in each TestFixture.
Test case dependencies What are the test cases whose successful outcome is a precondition for this to be executed? Will the running of one test case possibly interfere with the running of another test case? If yes, identify the cases and state the effects/resolution. For example running a stress test with reduced resources may leave the system in state that may interfere with other test cases.
Test case behaviour

26

Challenges of Software Testing



- We cannot test the whole system completely
 - To test all possible paths is IMPOSSIBLE!
 - To test all possible values is IMPOSSIBLE!
- Design effective test strategies
- You can't tell your system is bug-free!
 - Testing can reveal the presence of bugs, not their absences

27

Acknowledgement

The slides were originally authored by **Simon Ng**.



28