Software Testing

Reading: Testing Packet
Testing Introduction

- Testing is the process of finding software faults
  - Fault: “an incorrect step, process, or data definition in a program”
- Testing: “the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior”
- Test cases uncover failure by finding where the actual behavior of a program deviates from the expected behavior.
- Investigation of failures uncovering faults.
Why Test?

- Testing increases confidence that your program works correctly – meets the customer requirements
  - Better scores on assignments
  - Save companies millions, if not billions, of dollars

Verification vs. Validation

- Verification: process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase
  - Are we building the product right?
- Validation: process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements
  - Are we building right product?
Verification vs. Validation (2)

<table>
<thead>
<tr>
<th>Verification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Are we building the product <strong>right</strong>?</td>
<td>Are we building the <strong>right</strong> product?</td>
</tr>
</tbody>
</table>

```
“I landed on “Go” but didn’t get my $200!”
```

```
“I know this game has money and players and “Go” – but this is not the game I wanted.”
```

Black Box Testing

- Ignores the internals of the program – program treated as a black box
- Finds
  - Incorrect or missing function,
  - Interface errors,
  - Errors in data structures or external data base access,
  - Behavior or performance errors, and
  - Initialization and termination errors.
White Box Testing

- Code under test is known!
- Use code to guide test
- Your tests should exercise
  - Independent paths within the source code ("important" paths)
  - Logical decisions as both true and false
  - Loops at their boundaries
  - Internal data structures

Types of Testing

- Unit Testing
- Integration Testing
- Functional/System Testing
- Acceptance Testing
- Regression Testing
- Beta Testing
Unit Testing

- testing of individual hardware or software units or groups of related units
- Done by programmer(s)
- Generally all white box
- Automation desirable for repeatability

Integration Testing

- testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them
- Done by programmer as they integrate their code into code base
- Generally white box, maybe some black box
- Automation desirable for repeatability
Functional / System Testing

- testing conducted on a complete, integrated system to evaluate the system compliance with its specified requirements
- Stress testing, performance testing, usability testing
- It is recommended that this be done by external test group
- Mostly black box so that testing is not ‘corrupted’ by too much knowledge
- Test automation desirable

Acceptance Testing

- formal testing conducted to determine whether or not a system satisfies its acceptance criteria (the criteria the system must satisfy to be accepted by a customer) and to enable the customer to determine whether or not to accept the system
- Generally done by customer/customer representative in their environment through the GUI . . . Definitely black box
Regression Testing

- *Regression testing* is selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.

- Smoke test *group of test cases that establish that the system is stable and all major functionality is present and works under “normal” conditions*.

Beta Testing

- 3rd party testing by a subset of customers
- Unstructured/unscripted testing
- Black box
Development and Verification Activities

Level of Detail

Low

High

Project Time

Requirements Elicitation → Acceptance Testing

→ System Testing

→ Integration Testing

→ Unit Testing

→ Object Design

→ Design

→ Analysis

Requirements Elicitation

Testing Process

Write program

Write test cases

Run test on program

No Failure

Actual == Expected?

Yes

Deadline?

Yes

Deliver

No

Write more test cases

Pass
Test Case Information

- **Unique Identifier**
  - Black box: name of test in document
  - White box: name of test method

- **Input** into the program or program unit
  - Black box: how the user runs and interacts with the program
    - Could be redirection input and output like HW2
  - White box: inputs to methods that set up test

- **Expected output** from the program or program unit
  - What you expect to get based on input and requirements

- **Actual results** of running the test case
  - Black box: what the user gets from the program
    - Could be redirection of std out
  - White box: return values from functions or check on state via other functions

StringAnalyzer Problem

A client wants to collect information about the types of characters users use in a String input to a program. Specifically, the client wants to collect the number of times a digit between a minimum and maximum value, inclusive, is used in String input. The user specifies the string input, a minimum, and a maximum value. If the minimum and maximum values are not valid digits or if the maximum is less than the minimum, the error message, “Invalid bounds” is displayed to the user. Otherwise, the count of digits between the minimum and maximum values, inclusive, is displayed to the user in the following output: “The String contains X digits between minimum and maximum, inclusive.”

The program repeats until the user enters the string “quit”. 
Black Box Test Plan

- Formal document outlining the black box test cases for a project
- Description must be repeatable – have specific values!
- Expected results require specific values too!
- Write black box tests **before** writing your program

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Description</th>
<th>Expected Results</th>
<th>Actual Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestName</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Type:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test ID Description Expected Results Actual Results

Test Requirements

- Ensure that all of the customer requirements are tested!
Equivalence Classes

- Input space is broken into different classes
- Each equivalence class is tested
- Tests are written to include “middle” input values from each of the possible classes

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Description</th>
<th>Expected Results</th>
<th>Actual Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>noInput (Heckman)</td>
<td>Preconditions: StringAnalyzer program started and note that the string input for the first prompt is just pressing enter</td>
<td>The String contains 0 digits between 1 and 7, inclusive.</td>
<td></td>
</tr>
</tbody>
</table>

Boundary Value Analysis

- Programmers tend to make mistakes at boundaries
- Want to test program boundaries and values to either side of the boundary

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Description</th>
<th>Expected Results</th>
<th>Actual Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeroMinBound (Heckman)</td>
<td>Preconditions: StringAnalyzer program started</td>
<td>The String contains 3 digits between 0 and 7, inclusive.</td>
<td></td>
</tr>
</tbody>
</table>
Diabolical or Dirty Test Cases

- Divide by zero?
- Wrong input type? String instead of a double
- Illogical path through functionality
- Don’t enter mandatory fields?
- Program aborted or device unplugged, dropped, turned off?

Black Box Test Cases

- The input to a text field must be an integer between 5 and 10, inclusive. If the integer is outside the range, then an error message is displayed.

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Description</th>
<th>Expected Results</th>
<th>Actual Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
White Box Testing Techniques

- Basis set testing
  - Cyclomatic complexity $V(G)$ – estimate of number of paths
  - Test all program paths in a method
  - Test all conditionals on true AND false side
  - Control flow diagram
- Equivalence Classes
- Boundary Value Analysis
- Think diabolically

Control Flow Diagram

- Pictorial description of the flow of program control
- Diamonds: decisions
- Rectangles: program statements
- Break apart compound conditionals
- Loops have one decision (the continuation test)
  - Unless the loop has compound conditional tests – those should be broken up
Control Flow Templates

- for and while Loops
- do-while Loops
- if Statement

Sequential if Statements

Nested if/else/else Statements

Nested if/else Statements

if Statement

if/else Statement

if/else Statement

Compound Predicates Control Flow Templates

- predicate1 && predicate2
- predicate1 || predicate2
Exercise

- Create the control flow diagram for countDigits

```java
public int countDigits() {
    int digitCount = 0;
    for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        if (Character.isDigit(c)) {
            int v = Character.getNumericValue(c);
            if (v >= minValue && v <= maxValue) {
                digitCount++;
            }
        }
    }
    return digitCount;
}
```

StringAnalyzer.countDigits()
Cyclomatic Complexity

• Measure of a method’s complexity
• Number of potential paths through the source code

• Use to *estimate* number of tests to write
  – Some paths may not be possible
• Cyclomatic Complexity = # decisions (diamonds) + 1

Writing White Box Test Cases

• Focus on methods – how can we test?
  – Automation!
• Create a separate test class for each class that you need to test
• Exercise all paths of your class’ public or protected methods
  – Create control flow diagram
  – Inputs for basis set of paths
  – Inputs for equivalence classes and boundary values
  – Inputs for diabolical tests
JUnit for White Box Testing

- JUnit is an API for automating white box testing
  - Need to include JUnit libraries on classpath
  - Focus on testing paths through methods of the class under test
- Different versions of JUnit (we’re using JUnit 4)
  - Import
    - static org.junit.Assert.*;
    - org.junit.Before;
    - org.junit.Test;
  - Test case methods have @Test tag before each test method
  - setUp() automatically called before EACH test method
    - @Before tag required

Assert Methods

- Assert methods provide information about the expected and actual values of a test case
  - assertEquals(expected, actual);
    - For doubles, you will have a third argument, delta
  - assertTrue(actual);
  - assertFalse(actual);
  - assertNull(actual);
  - assertNotNull(actual);
- Could also include a leading parameter containing a message (id)
Example JUnit Program

```java
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;
public class StringAnalyzerTest {
    @Test
    public void testCountDigits1() {
        //Test path 1-2-9
        StringAnalyzer st = new StringAnalyzer("", 3, 7);
        assertEquals(0, st.countDigits());
    }
}
```

White Box Testing Techniques

- Basis set testing
  - Cyclomatic complexity $V(G)$ – estimate of number of paths
  - Test all program paths in a method
  - Test all conditionals on true AND false side
  - Control flow diagram
- Equivalence Classes
- Boundary Value Analysis
- Think diabolically
Failing Tests

- If a test fails, use the debugger to find and fix the problem.

References

- L. Williams, CSC326 Slides