Software testing
February 25, 2020

Today

Introduction to software testing

- Blackbox vs. whitebox testing
- Unit testing (vs. integration vs. system testing)
- Test adequacy
  - Structural code coverage
    - Statement coverage
    - Decision coverage
    - Condition coverage
  - Mutation analysis

Software testing

What can testing do, and what can’t it do?

Software testing can show the presence of defects, but never show their absence! (Edsger W. Dijkstra)

- A good test is one that fails because of a defect.

Thursday (February 27)

- Second in-class exercise
- On testing (today is a prelude with useful info)
- Form 4-person teams
  - Use moodle to self-select a team; can do it before Thursday or on Thursday.
  - At least one person per team needs to bring a laptop
    BRING A LAPTOP!
Two strategies: black box vs. white box

Black box testing
- The system is a black box (can't see inside).
- No knowledge about the internals of a system.
- Create tests solely based on the specification (e.g., input/output behavior).

White box testing
- Knowledge about the internals of a system.
- Create tests based on these internals (e.g., exercise a particular part or path of the system).

Unit testing, integration testing, system testing

Unit testing
- Does each unit work as specified?

Integration testing
- Do the units work when put together?

System testing
- Does the system work as a whole?

Our focus: unit testing

Unit testing
- A unit is the smallest testable part of the software system.
- Goal: Verify that each software unit performs as specified.
- Focus:
  - Individual units (not the interactions between units).
  - Usually input/output relationships.

JUnit 4: Overview
- Provides the xUnit testing framework for Java
- Uses Java annotations to specify tests and test suites
JUnit 4: Sample Rectangle class constructor

```java
public class Rectangle {
    int width, int height;
    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }
}
```

Available from here: https://github.com/LASER-UMASS/cs520-Spring2020.git
(Contained in the rectangle folder)

JUnit 4: Test – Normative behavior

A single unit test [@Test]

```java
@Test
public void testNonRectangleSatisfiesPrecondition() {
    // Given inputs
    int width = -1;
    int height = 0;
    // Test on these inputs
    assertEquals(width, rectangle.getWidth());
    assertEquals(height, rectangle.getHeight());
}
```

JUnit 4: Test – Exceptional behavior

A fixed set of objects used as a baseline to run tests (to be able to replicate test results) can be run before/after:

- each method (test) [@Before/After]
- each class (often test suite) [@BeforeClass/AfterClass]

```java
@Before
public void setUp() {
    rectangle = new Rectangle(WIDTH, HEIGHT);
    square = new Rectangle(WIDTH, WIDTH);
}
```

```java
@After
public void tearDown() {
    rectangle = null;
    square = null;
}
```
JUnit 4: Test –
Normative behavior (using a text fixture)

@Test
public void testNewRectangleSatisfiesPrecondition() {
    // Given known inputs
    Assert.assertTrue(WIDTH > 0);
    Assert.assertNotNull(rectangle);
    Assert.assertEquals(WIDTH, rectangle.getWidth());
    Assert.assertEquals(HEIGHT, rectangle.getHeight());
}

@Test
public void testEqualNullReturnsFalse() {
    Assert.assertNull(rectangle);
    Rectangle nullRectangle = null;
    Assert.assertEquals(null, nullRectangle);
    Assert.assertFalse(rectangle.equals(nullRectangle));
}

JUnit 4: Test suite - Implied

A test suite consists of one or more tests or test suites

JUnit 4: Rectangle class other methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean equals(java.lang.Object obj)</td>
<td>Gets the area of this rectangle.</td>
</tr>
<tr>
<td>int getArea()</td>
<td>Gets the width of this rectangle.</td>
</tr>
<tr>
<td>int getLength()</td>
<td>Gets the height of this rectangle.</td>
</tr>
<tr>
<td>boolean isSquare()</td>
<td>Returns true if this rectangle is a square and false otherwise.</td>
</tr>
<tr>
<td>void setHeight(int height)</td>
<td>Sets the height of this rectangle to the given positive number.</td>
</tr>
<tr>
<td>void setWidth(int width)</td>
<td>Sets the width of this rectangle to the given positive number.</td>
</tr>
</tbody>
</table>

JUnit 4: Test suite - Explicit

```java
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@SuiteSuitableClass{}
public class RectangleTestSuite {
    // the class contains empty,
    // used only as a holder for the above annotations
}```
JUnit 4: Running a test suite

From command line:
- “java org.junit.runner.JUnitCore -cp ../lib/junit-4.11.jar:bin RectangleTests”

From Eclipse UI:
1. Right click on RectangleTests.java
2. In the context menu, select “Run As > JUnit Test”

NOTE) Can replace RectangleTests.java with RectangleTestSuite.java in the above

JUnit 4: Interpreting the test suite results

JUnit 4: Interpreting the test suite results
Software testing

Software testing can show the presence of defects, but never show their absence! (Edsger W. Dijkstra)

- A good test is one that fails because of a defect.

When should we stop testing if no (new) test fails?

Test effectiveness

Ratio of detected defects is the best effectiveness metric!

Problem
- The set of defects is unknowable

Solution
- Use a proxy metric, for example code coverage

---

Structural code coverage: live example

Average of the absolute values of an array of doubles

```java
public double avgAbs(double ... numbers) {
    // we expect the array to be non-null and non-empty
    if (numbers == null || numbers.length == 0) {
        throw new IllegalArgumentException("Array numbers must not be null or empty!");
    }
    double sum = 0;
    for (int i=0; i<numbers.length; i++) {
        double d = numbers[i];
        if (d < 0) {
            sum -= d;
        } else {
            sum += d;
        }
    }
    return sum / numbers.length;
}
```

Control Flow Graph (CFG)

![Control Flow Graph](image-url)
Statement coverage

- **Every statement** in the program must be executed at least once
- Given the control-flow graph (CFG), this is equivalent to node coverage

Condition coverage vs. decision coverage

**Terminology**

- **Condition**: a boolean expression that cannot be decomposed into simpler boolean expressions.
- **Decision**: a boolean expression that is composed of conditions, using 0 or more logical connectors (a decision with 0 logical connectors is a condition).
- **Example**: if \((x<5) \&\& (y>7))\) { ... }
  - \((x<5)\) and \((y>7)\) are conditions.
  - The boolean expression \((x<5) \&\& (y>7))\) is a decision.

Decision coverage (a.k.a. branch coverage)

- **Every decision** in the program must take on all possible outcomes (true/false) at least once
- Given the CFG, this is equivalent to edge coverage
- **Example**: if \((a>0 \&\& b>0)\)
  - \(a=1, b=1\)
  - \(a=0, b=0\)
**Condition coverage**

- Every condition in the program must take on all possible outcomes (true/false) at least once
- Example: \((a>0 \&\& b>0)\)
  - \(a=1, b=0\)
  - \(a=0, b=1\)

**Subsumption**

Given two coverage criteria A and B, A subsumes B if satisfying A implies satisfying B

- Subsumption relationships:
  - Does decision coverage subsume statement coverage?
  - Does decision coverage subsume condition coverage?
  - Does condition coverage subsume decision coverage?
Decision coverage vs. condition coverage

4 possible tests for the decision $a \lor b$:

1. $a = 0, b = 0$
2. $a = 0, b = 1$
3. $a = 1, b = 0$
4. $a = 1, b = 1$

- Satisfies condition coverage but not decision coverage
- Satisfies decision coverage but not condition coverage

Neither coverage criterion subsumes the other!

Structural code coverage: subsumption

Given two coverage criteria $A$ and $B$,

$A$ subsumes $B$ if satisfying $A$ implies satisfying $B$

- Subsumption relationships:
  - Decision coverage subsumes statement coverage
  - Decision coverage does not subsume condition coverage
  - Condition coverage does not subsume decision coverage

Code coverage: advantages

- Code coverage is easy to compute.
- Code coverage has an intuitive interpretation.

But, does coverage ensure effective testing?

Code coverage: drawbacks

- Code coverage does not require test assertions.
- Not all statements etc. are equally important.
- Coverage is not the same as behavior.

Are there any alternatives?
Mutation analysis: overview

- Program
- Test suite
- Generate mutants
- Mutants

Each mutant contains one small syntactic change.
Mutation analysis: overview

Assumption: Mutant detection rate is a good proxy for fault detection rate.

What does it mean for a test to fail on a mutant program?

Mutation analysis: example

Find a test case that detects the following mutant
(i.e., passes on the original program but fails on the mutant)

Original program:
```java
public int min(int a, int b) {
    return a < b ? a : b;
}
```

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>Original</th>
<th>Mutant</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Mutant:
```java
public int min(int a, int b) {
    return a;
}
```
Mutation analysis: another example

Find a test case that detects the following mutant (i.e., passes on the original program but fails on the mutant)

Original program:
```java
public int min(int a, int b) {
    return a < b ? a : b;
}
```

Mutant:
```java
public int min(int a, int b) {
    return a <= b ? a : b;
}
```

There is no such test that can detect the mutant...
The mutant is undetectable because it is equivalent to the original program!

Summary

- Testing is an important way to measure code quality
- Black-box testing
- White-box testing
- Coverage metrics
  - Statement
  - Condition
  - Decision
- Mutation-based metric

For more, read:
"Are mutants a valid substitute for real faults in software testing?" in FSE 2014

Final project

https://people.cs.umass.edu/~hconboy/class/2020Spring/CS520/finalProject.pdf

- Four choices:
  - MSR 2020 mining challenge
  - Replication study
  - Model inference for inferring processes (i.e., specification mining)
  - EleNa: Elevation-based Navigation

Group and Project selection due Tue, Mar 3