CS 520
Theory and Practice of Software Engineering
Fall 2022

In-class exercise 2: Software testing

October 13, 2022
Roadmap

• Unit testing
• Test effectiveness
  – Code coverage
  – Mutation analysis
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: Test – Normative behavior

A single unit test [@Test]

```java
@Test
public void testNewRectangleSatisfiesPrecondition() {
    // Given known inputs
    Assert.assertTrue(WIDTH > 0);
    Assert.assertTrue(HEIGHT > 0);
    // Test on those inputs
    Rectangle rectangle = new Rectangle(WIDTH, HEIGHT);
    // Check for expected output
    Assert.assertNotBeNull(rectangle);
    Assert.assertEquals(WIDTH, rectangle.getWidth());
    Assert.assertEquals(HEIGHT, rectangle.getHeight());
}
```
JUnit 4: Test – Exceptional behavior

```java
// Check expected output
@Test(expected=IllegalArgumentException.class)
public void testNewRectangleViolatesPrecondition() {
    // Given known inputs
    int negativeWidth = - WIDTH;
    Assert.assertFalse(negativeWidth > 0);
    Assert.assertTrue(HEIGHT > 0);
    // Test on those inputs
    Rectangle newRectangle = new Rectangle(negativeWidth, HEIGHT);
}
```
JUnit 4: Running test suite(s) with ant

```xml
<!-- Execute the test suite -->
<target name="test" depends="compile.tests" description="Run all unit tests">
  <echo message="Running unit tests ..."/>
  <junit printsummary="true"
        showoutput="true"
        haltonfailure="false">
    <formatter type="plain" usefile="false"/>
    <classpath path="bin"/>
    <classpath location="${junit.jar}"/>

    <batchtest fork="no">
      <fileset dir="test">
        <include name="**/*.java"/>
      </fileset>
    </batchtest>
  </junit>
</target>
```
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 or mutation detection rate
Statement coverage (a.k.a. line 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 (a.k.a. branch coverage)

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

**NOTE** For in-class exercise 2, the coverage tool refers to condition coverage as branch coverage.

*In general, decision coverage is usually referred to as branch coverage.*
Mutation analysis

Assumption: Mutant detection rate is a good proxy for fault detection rate.
Mutation analysis: first 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;
}
```

<table>
<thead>
<tr>
<th>a</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>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Mutation analysis: second 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;
}
```
Mutation analysis: second 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;
}
```

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Mutant</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Mutation analysis: second 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!
Mutant detection rate

Search for a test case that passes on the original program but fails on the mutant

• If found, the mutant is not equivalent to the original program. This is called a detectable mutant.
• If not found, the mutant is equivalent to the original program. This is called an undetectable mutant.

\[
\text{mutant detection rate} = \frac{\# \text{ detectable mutants}}{\# \text{ all mutants}}
\]
Group selection

• Form 2-, 3-, or 4-person teams
  ○ If you need more members in your team, raise your hand and ask the instructor
• Use Moodle to self-select a team

• Select a team member responsible for submitting the completed exercise by next Thursday
Set up

1. Make sure that you have Git (v2.7.4 or later) and Java (v8 or later) installed.
- Git: https://git-scm.com/

USEFUL TIP: Make sure that the JDK comes early in the PATH environment variable. This can fix a lot of issues with modern and multiple JDK installations. For more information, see https://www.java.com/en/download/help/path.xml

2. Clone the cs520 git repository:
git clone https://github.com/LASER-UMASS/cs520 inclass2