Intro to software testing

October 17, 2017
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

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

How do we come up with good tests?
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.
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;
}
```

Executable example available on the course web site.
Structural code coverage: live example

Entry point

a == null || a.length == 0

sum = 0
i = 0

i < a.length

num = a[i]

num < 0

sum += num
sum -= num
++i

true
false
false
true
false
true

return sum / a.length

throw new IllegalArgumentException("Array a must not be null or empty!")

Exceptional exit
Normal exit
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
Structural code coverage: statement coverage

Entry point

\(a==null \text{ || } a.length==0\)

false

sum = 0
i = 0

\(i<a.length\)

false

return sum/a.length

true

true

throw new IllegalArgumentException("Array a must not be null or empty!")

false

Normal exit

Exceptional exit

i < a.length

true

false

false

false

true

num = a[i]

num < 0

true

false

sum += num

sum -= num

++i
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 \((a \&\& b)\) { ... }
  - \(a\) and \(b\) are *conditions*.
  - The boolean expression \(a \&\& b\) is a *decision*. 
Decision coverage (aka 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:** \((a>0 \& b>0)\)
  - \(a=1, b=1\)
  - \(a=0, b=0\)
Structural code coverage: decision coverage

```
Entry point

a==null || a.length==0
true
throw new IllegalArgumentException("Array a must not be null or empty!")
false

sum = 0
i = 0

i<a.length
false
return sum/a.length
true
num = a[i]

num < 0
false
sum += num
true
sum -= num
```

++i
Condition coverage

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

\[ a == \text{null} \lor a\text{.length} == 0 \] if true, throw new IllegalArgumentException("Array a must not be null or empty!")

\[ \text{sum} = 0 \]
\[ i = 0 \]

\[ i < a\text{.length} \] if false, return \( \text{sum}/a\text{.length} \)

\[ \text{num} = a[i] \]

\[ \text{num} < 0 \] if false, \( \text{sum} += \text{num} \)

\[ \text{num} = a[i] \]

\[ \text{sum} -= \text{num} \]

\[ ++i \]
Structural code coverage: subsumption

Given two coverage criteria A and B, A subsumes B iff 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 $a \mid b$:

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

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$a \mid b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Satisfies **condition coverage** but not **decision coverage**

Does not satisfy **condition coverage** but **decision coverage**

Neither coverage criterion subsumes the other!
Structural code coverage: subsumption

Given two coverage criteria A and B, A subsumes B iff 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.
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.

Are there any alternatives?
Mutation analysis: overview

Program

Test suite
Mutation analysis: overview
Mutation analysis: overview

```
public float avg(float[] data) {
    float sum = 0;
    for (float num : data) {
        sum += num;
    }
    return sum / data.length;
}
```

```
public float avg(float[] data) {
    float sum = 1;
    for (float num : data) {
        sum += num;
    }
    return sum / data.length;
}
```

Each mutant contains one small syntactic change
Mutation analysis: overview

```java
public float avg(float[] data) {
    float sum = 0;
    for (float num : data) {
        sum += num;
    }
    return sum / data.length;
}
```

```java
public float avg(float[] data) {
    float sum = 0;
    for (float num : data) {
        sum += num;
    }
    return sum / data.length;
}
```
Mutation analysis: overview

Program

Generate mutants

Mutants

public float avg(float[] data) {
    float sum = 0;
    for (float num : data) {
        sum += num;
    }
    return sum / data.length;
}

public float avg(float[] data) {
    float sum = 0;
    for (float num : data) {
        sum += num;
    }
} return sum * data.length;

public float avg(float[] data) {
    float sum = 0;
    for (float num : data) {
        sum += num;
    }
} return sum / data.length;
Mutation analysis: overview

Program

Generate mutants

Mutants

Test suite

Execute test suite

Mutant detection rate
Mutation analysis: overview

Assumption: Mutant detection rate is a good proxy for fault detection rate.
(https://people.cs.umass.edu/~rjust/publ/mutants_real_faults_fse_2014.pdf)
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:
public int min(int a, int b) {
    return a < b ? a : b;
}

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

Mutant:
public int min(int a, int b) {
    return a;
}

<table>
<thead>
<tr>
<th></th>
<th></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: 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:
public int min(int a, int b) {
    return a < b ? a : b;
}

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

Mutant:
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!
Mutation analysis: 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;
}
```

Executable example available on the course web site.