Course projects

1. Review proposed projects
   - Test case/scenario generation for Robotics (4)
   - Smarter code completion (2)
   - E-graphs for equivalent/redundant mutant detection (4)

2. Clarify open questions

3. Form groups

Software Testing 101

Today

- Course projects
- Introduction to software testing
  - Blackbox vs. whitebox testing
  - Unit testing (vs. integration vs. system testing)
  - Test adequacy: code coverage
    - Statement coverage
    - Decision coverage (Branch coverage)
    - Condition coverage
    - Path coverage
- Discussion of DART: Directed Automated Random Testing
Software testing vs. software debugging

Testing: is there a bug?

```java
double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    int i = 0;
    while (i<n) {
        sum = sum + nums[i];
        i = i + 1;
    }
    double avg = sum * n;
    return avg;
}
```

Testing: is there a bug?

```java
@Test
public void testAvg() {
    double[] nums = new double[]{1.0, 2.0, 3.0};
    double actual = Math.avg(nums);
    double expected = 2.0;
    assertEquals(expected, actual, EPS);
}
```

testAvg failed: 2.0 != 18.0

Debugging: where is the bug?

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?

---

### Unit testing

- A **unit** is the **smallest testable part** of the software system (e.g., a method in a Java class).
- **Goal**: Verify that each software unit performs as specified.
- **Focus**:
  - Individual units (not the interactions between units).
  - Usually input/output relationships.

---

### Test effectiveness

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.

---

### 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.

---

*How do we come up with good tests?*
Average of the absolute values of an array of doubles

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

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 \((a \& b)\) { … }
  - \(a\) 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**: \((a>0 \& b>0)\)
  - \(a=1, b=1\)
  - \(a=0, b=0\)

Decision coverage (aka branch coverage)

Condition coverage

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

Structural code coverage: subsumption

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

- Subsumption relationships:
  - Does statement coverage subsume decision coverage?
  - 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>( a )</th>
<th>( b )</th>
<th>( a \mid b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<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!

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>( a )</th>
<th>( b )</th>
<th>( a \mid b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<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
Path coverage

Entry point

\[
a == \text{null} || \ a.length == 0
\]

\[
\text{false}
\]

sum = 0
i = 0

\[
i < a.length
\]

\[
\text{false}
\]

return sum / a.length

Exceptional exit

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

Normal exit

true
false

num = a[i]

true
false

num < 0
sum += num

false
true

sum -= num
++i
num = a[i]

DART: Directed Automated Random Testing