Example: Single slot buffer requirements (specified in Natural Language)

1. The initial size is 0. The maximum size is 1. The $\text{size()}$ method returns either 0 or 1.
2. The single element has type $E$.
3. The $\text{write(element : } E\text{)}$ method attempts to store the given element into the single slot buffer. This method returns void when the size is 0. It throws a BufferOverflow exception when the size is 1.
4. The $\text{read()}$ method attempts to retrieve the stored element from the single slot buffer. This method returns that element when the size is 1. It throws a BufferUnderflow exception when the size is 0.

Example: Single slot buffer requirement (specified as an FSA alphabet)

- size(): 0
- size(): 1
- write(E): void
- write(E): BufferOverflow
- read(): E
- read(): BufferUnderflow

**NOTE:** In the following, we’ll use “*” to mean all events in the alphabet.
Example: Single slot buffer requirement (specified as an FSA)

- Size: 0,
  - Read(): BufferUnderflow

Start state: 0
Accepting states: {0, 1}

- Size: 1,
  - Write(E): BufferOverflow,
  - Read(): E

Example: Single slot buffer requirement (specified as an FSA)

- Size: 0,
  - Read(): BufferUnderflow

- Size: 1,
  - Write(E): BufferOverflow,
  - Read(): E

- Read(): E
  - Size: 1,
  - Write(): BufferOverflow
  - Read(): BufferUnderflow

- Size: 0,
  - Write(E): void

- Size: 1,
  - Write(E): void

- Read(): E
  - Size: 1,
  - Write(E): void,
  - Read(): E

- Read(): E
  - Size: 1,
  - Write(E): BufferOverflow,
  - Read(): E

- Size: 0,
  - Write(E): void,
  - Read(): BufferUnderflow

- Size: 1,
  - Write(E): void,
  - Read(): BufferUnderflow

- Read(): E
Example: Single slot buffer requirement (specified as an FSA)

<table>
<thead>
<tr>
<th>State</th>
<th>Transition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>read(): E</td>
<td>BufferUnderflow</td>
</tr>
<tr>
<td>0</td>
<td>write(E)</td>
<td>void</td>
</tr>
<tr>
<td>1</td>
<td>read(): E</td>
<td>BufferOverflow</td>
</tr>
<tr>
<td>1</td>
<td>write():</td>
<td>BufferOverflow</td>
</tr>
</tbody>
</table>


NOTE) The violation state and its transition are now implied.

Are the following event sequences accepted or rejected?

- []
- [size: 1]
- [read(): E]
- [write(E): void, read(): E]
Thursday (October 1)

- Second in-class exercise
- On testing (today is a prelude with useful info)
- Form 3-, 4-, or 5- person teams
  - Use Moodle to self-select a team; open from today until Friday at noon
  - After closing, the remaining students will be randomly assigned to groups
  - In your group’s description, let us know if your group wants us to set up a Zoom breakout room for Thursday

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
  - User acceptance testing

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.

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**

**JUnit**

**JUnit 4: Overview**
- Provides the xUnit testing framework for Java
- Uses Java annotations to specify tests and test suites
- More details available here:

**JUnit 4: Sample Rectangle class constructor**

```
public Rectangle(int width, int height) {
    if (width < 0 || height < 0) {
        throw new IllegalArgumentException()
    }
}
```

Available from here:
(Contained in the rectangle folder)
JUnit 4: Test – Normative behavior

**A single unit test [@Test]**

```java
@Test
public void testNewRectangleSatisfiesPrecondition() {
    // Given known input
    int width = 10;
    int height = 20;

    // Test on those inputs
    Rectangle rectangle = new Rectangle(width, height);
    // Check for expected output
    assertEquals(10, rectangle.getWidth());
    assertEquals(20, rectangle.getHeight());
}
```

JUnit 4: Test – Exceptional behavior

**Normative behavior (using a text fixture)**

```java
@Test
public void testNewRectangleSatisfiesPrecondition() {
    // Given known inputs
    int width = 10;
    int height = 20;

    // Test on those inputs
    Rectangle rectangle = new Rectangle(width, height);
    assertEquals(10, rectangle.getWidth());
    assertEquals(20, rectangle.getHeight());
}
```

JUnit 4: Test fixture

**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;
}
```

```java
@Test
public void testEqualsNullReturnsFalse() {
    Rectangle nullRectangle = null;
    assertEquals(false, nullRectangle.equals(nullRectangle));
}
```
JUnit 4: Rectangle class other methods

```java
boolean equals(java.lang.Object object)
int getArea()
int getHeight()
int getWidth()
int hashCode()
boolean isSquare()

void setHeight(int height)
void setWidth(int width)
```

JUnit 4: Test suite - Implied

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

JUnit 4: Test suite - Explicit

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

@RunWith(Suite.class)
@Suite.SuiteClasses(RectangleTests.class)

public class RectangleTestSuite {
    // the class remains 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
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)

- Every statement in the program must be executed at least once
- Given the control-flow graph (CFG), this is equivalent to node coverage
Statement 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:** \((a>0 \&\& b>0)\)
  - \(a=1, b=0\)
  - \(a=0, b=1\)

---

**Result coverage**

Entry point

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

Normal exit

Exceptional exit

- \(\text{throw new IllegalArgumentException(“Array a must not be null or empty!”)}\)

---

**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 \(a \| b\):

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 | b)</th>
<th>(a)</th>
<th>(b)</th>
<th>(a | b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- 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 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.

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
Mutation analysis: overview

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

Each mutant contains one small syntactic change

```
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

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>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>
</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;
}
```

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

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

User acceptance testing

- **Alpha testing**: The development team releases alpha versions of the UI to internal users in a simulated setting.
- **Beta testing**: The development team releases beta versions of the UI to target an external user group in a real-world setting.

The alpha and beta testers submit bug reports and provide their feedback on using the UI.

https://www.webopedia.com/Term/B/beta_test.html

Commonly use the SUS survey for feedback

The System Usability Scale

<table>
<thead>
<tr>
<th>The System Usability Scale</th>
<th>Strongly disagree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 I think that I would like to use this system.</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>2 I found the system unnecessarily complex.</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>3 I thought the system was easy to use.</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>4 I think that I would need the support of a technical person to be able to use this system.</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>5 I found the various functions in the system were well integrated.</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>6 I thought the system was too complicated.</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>7 I found the system very stressful.</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>8 I found the system very user-friendly.</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>9 I felt very confident using the system.</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>10 I needed to learn a lot of things before I could get going with this system.</td>
<td>⬤</td>
<td>⬤</td>
</tr>
</tbody>
</table>