SWE 637 Software Testing

Activities, week 3

Unit Testing with JUnit

Dr. Brittany Johnson-Matthews
(Dr. B for short)

https://go.gmu.edu/SWE637

Adapted from slides by Jeff Offutt and Bob Kurtz

Class Activity #3

Consider the `Point` class

- What should the implementation of `equals()` look like?
- Develop some JUnit tests for `equals()`
- Develop some parameterized (data-driven) JUnit tests for `equals()`
- Develop some JUnit theories about `equals()`
  - hint: overriding `equals()` means you must override `hashCode()` also

```
class Point
{
   private int x;
   private int y;

   public Point(int x, int y)
   {
      this.x=x;
      this.y=y;
   }

   @Override public boolean equals(Object o)
   {
      // What should the implementation be?
   }
}
```

Focus on what you want to test, not the JUnit syntax

Class Activity #3 - Answers

Possible implementation of `equals()`

```
class Point
{
   private int x;
   private int y;

   public Point(int x, int y)
   {
      this.x=x;
      this.y=y;
   }

   @Override public boolean equals(Object o)
   {
      if (!(o instanceof Point))
      {
         return false;
      }
      else
      {
         Point p = (Point) o;
         return (p.x == this.x) && (p.y == this.y);
   }
}
```

Class Activity #3 - Answers

JUnit tests for `Point.equals()`

```
public class PointTest
{
    @Test
    public void testEquals()
    {
        Point p1 = new Point (1, 2);
        Point p2 = new Point (1, 2);
        Point p3 = new Point (-1, 99);

        assertTrue (p1.equals(p1));
        assertTrue (p1.equals(p2));
        assertFalse (p1.equals(p3));

        assertTrue (p2.equals(p1));
        assertTrue (p2.equals(p2));
        assertFalse (p2.equals(p3));

        assertFalse (p3.equals(p1));
        assertFalse (p3.equals(p2));
        assertTrue (p3.equals(p3));
}
```

Class Activity #3 - Answers

Parameterized tests for `Point.equals()`

```
@RunWith(Parameterized.class)
public class PointParameterizedTest
{
    // Define test inputs
    public int x1, y1, x2, y2;

    // Define expected output
    public boolean isEqual;

    // Create a constructor to set up the
    //  parameterized data
    public PointParameterizedTest(int x1, int y1,
      int x2, int y2, boolean isEqual)
    {
        this.x1 = x1;
        this.y1 = y1;
        this.x2 = x2;
        this.y2 = y2;
        this.isEqual = isEqual;
    }
```

```
    @Parameters
    public static Collection<Object>[] params()
    {
        return Arrays.asList(new Object[][] {
            { 1, 2, 1, 2, true },
            { 1, 2, -1, 99, false },
            { -1, 99, -1, 99, true },
            { -1, 99, 1, 2, false }
        });
    }

    @Test
    public void testEquals()
    {
        Point p1 = new Point (x1, y1);
        Point p2 = new Point (x2, y2);
        assertEquals (isEqual, p1.equals(p2));
        assertEquals (isEqual, p2.equals(p1));
    }
}
```

SWE 637 Software Testing

System Testing with Cucumber

Dr. Brittany Johnson-Matthews
(Dr. B for short)

https://go.gmu.edu/SWE637

Adapted from slides by Jeff Offutt and Bob Kurtz

Boeing 737 MAX MCAS System

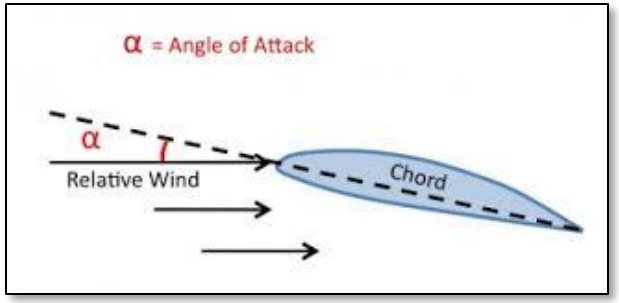
Maneuvering Characteristics Augmentation System (MCAS)

Automatic system intended to prevent excessive nose-up aircraft attitude which can lead to aerodynamic stall

Boeing 737 MAX MCAS System

MCAS takes 3 inputs:

- **Autopilot status (on/off)**
  - MCAS is only active when the autopilot is **off** and the pilot is hand-flying the aircraft
- **Flaps position (up/down)**
  - When lowered, flaps allow the aircraft to fly slower
  - MCAS is only active when flaps are **up**
- **Angle of attack (AOA)**
  - Angle of the wing relative to the airflow
  - Wing will stall (stop generating lift) if the AOA is too high
  - MCAS activates when AOA is **high** and activates the electric trim system to push the aircraft nose down to reduce AOA



Boeing 737 MAX MCAS System

Measuring AOA

- The 737 has one AOA vane on each side of the nose
- MCAS (in 2018/2019) used ***only*** the pilot's side AOA vane

AOA vane troubles

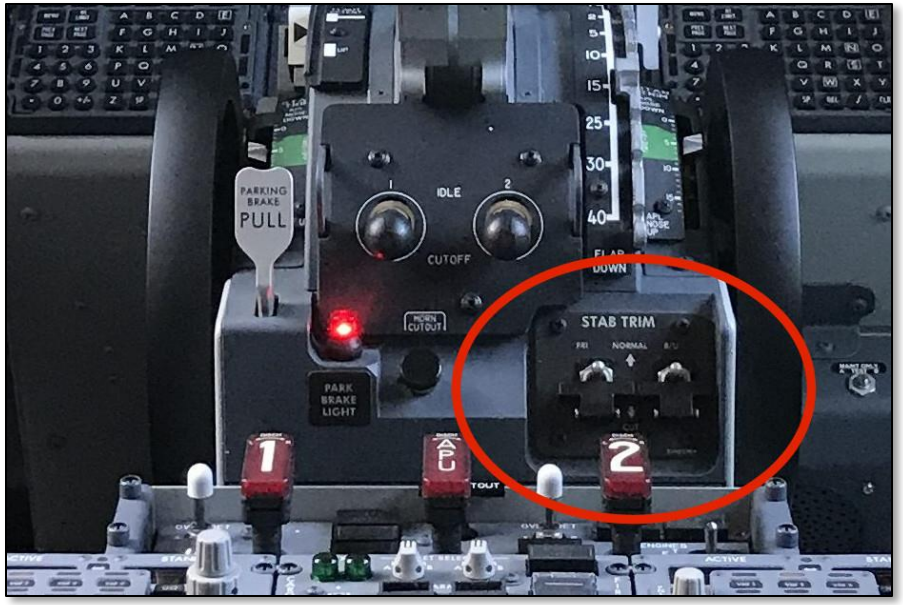
- On the Lion Air flight, the AOA vane had not been properly calibrated after replacement
- On the Ethiopian Airlines flight, it is likely that a bird strike during takeoff damaged the AOA vane
- Both aircraft thought the AOA was too high

Boeing 737 MAX MCAS System

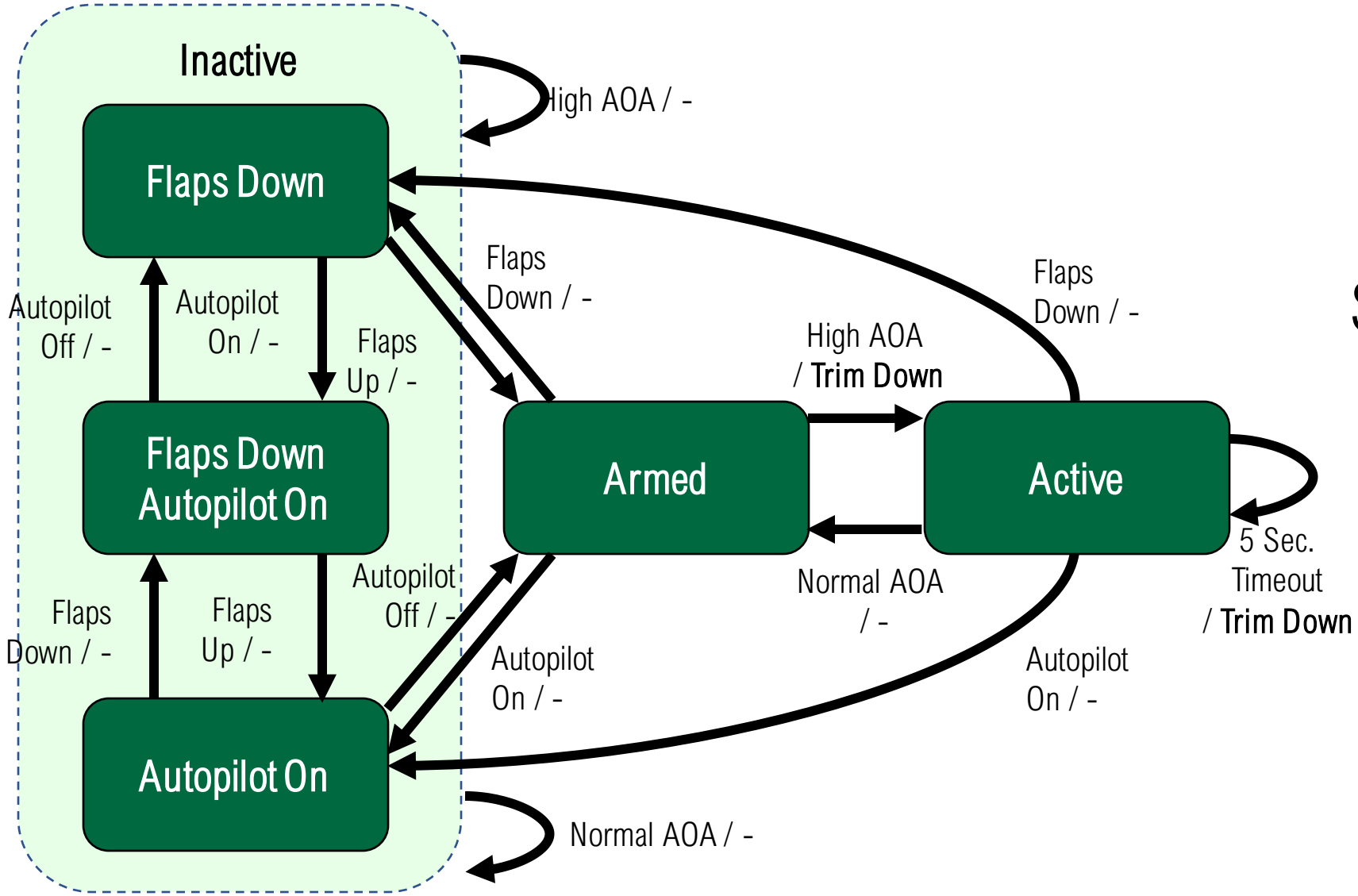
AOA vane failures and trim system failures happen, and they're part of flight training

MCAS can be disabled by flipping off the trim switches

- The Lion Air pilots never disabled the trim system
- The Ethiopian Airlines pilots *did* disable the trim system, but *then re-enabled it*



Boeing 737 MAX MCAS System



Simplified MCAS state diagram

Testing MCAS with Gherkin

Using the Gherkin system test language, design a system test to verify that MCAS activates (that is, produces a trim-down input) as desired

```
Scenario: McasActivates
Given …
When …
Then …
```

Testing MCAS with Gherkin

```
Scenario: McasActivates
Given flaps are up
And autopilot is off
When AOA is high
Then MCAS trims nose down
And MCAS delays for 5 seconds
```

Testing MCAS with Gherkin

Using the Gherkin system test language, design system tests to verify that MCAS does not activate when it should not

1. When flaps are down
2. When auto-pilot is on
3. When AOA is normal

Testing MCAS with Gherkin

```
Scenario: McasDoesNotActivate
Given …
When …
Then …

Scenario: McasDoesNotActivate
Given …
When …
Then …

Scenario: McasDoesNotActivate
Given …
When …
Then …
```

Testing MCAS with Gherkin

```
Scenario: McasNoActivateWhenFlapsDown
Given flaps are down
And autopilot is off
When AOA is high
Then MCAS does nothing

Scenario: McasNoActivateWhenAutopilotOn
Given flaps are up
And autopilot is on
When AOA is high
Then MCAS does nothing

Scenario: McasNoActivateWhenAoaNormal
Given flaps are up
And autopilot is off
When AOA is normal
Then MCAS does nothing
```