# Software Testing Training

Aluno: Altieres de Matos
altitdb@gmail.com

Orientador: Prof. Dr. Marco Aurélio Graciotto Silva
Coorientador: Prof. Dr. Reginaldo Ré

## Summary

- Software Testing
- Test Criteria
  - Equivalence Class Testing
  - Boundary Value Analysis
- Strategy Development
  - Test Addition, Test Last, Test First, Refactoring and Test Driven Development
- Bowling Game
  - Training
  - Microservices

## Software Testing

*"Testing is the process of executing a program with the intent of finding errors."*

The Art of Software Testing, 3rd Edition.
GLENFORD J. MYERS
TOM BADGETT
COREY SANDLER

## Test Criteria

- Equivalence Class Testing
- Boundary Value Analysis

## Equivalence Class Testing

- Identifying the equivalence classes
- Defining the test cases

## Identifying Equivalence Classes

- R3. The player is entitled to two shots to reach the maximum score (10) on each round.

| **External Condition** | **Valid Equivalence Classes** | **Invalid Equivalence Classes** |
|---|---|---|
| Maximum score | 0 <= score <= 10 | score < 0 or score > 10 |

# Defining Test Cases

- R3. The player is entitled to two shots to reach the maximum score (10) on each round.

| **Test Case** | **Result** |
|---|---|
| Score (-5) | Invalid value |
| Score (13) | Invalid value |
| Score (8) | Score saved |

## Boundary Value Analysis

Boundary value analysis differs from equivalence partitioning in two respects:

1. Rather than selecting any element in an equivalence class as being representative, boundary value analysis requires that one or more elements be selected such that each edge of the equivalence class is the subject of a test.

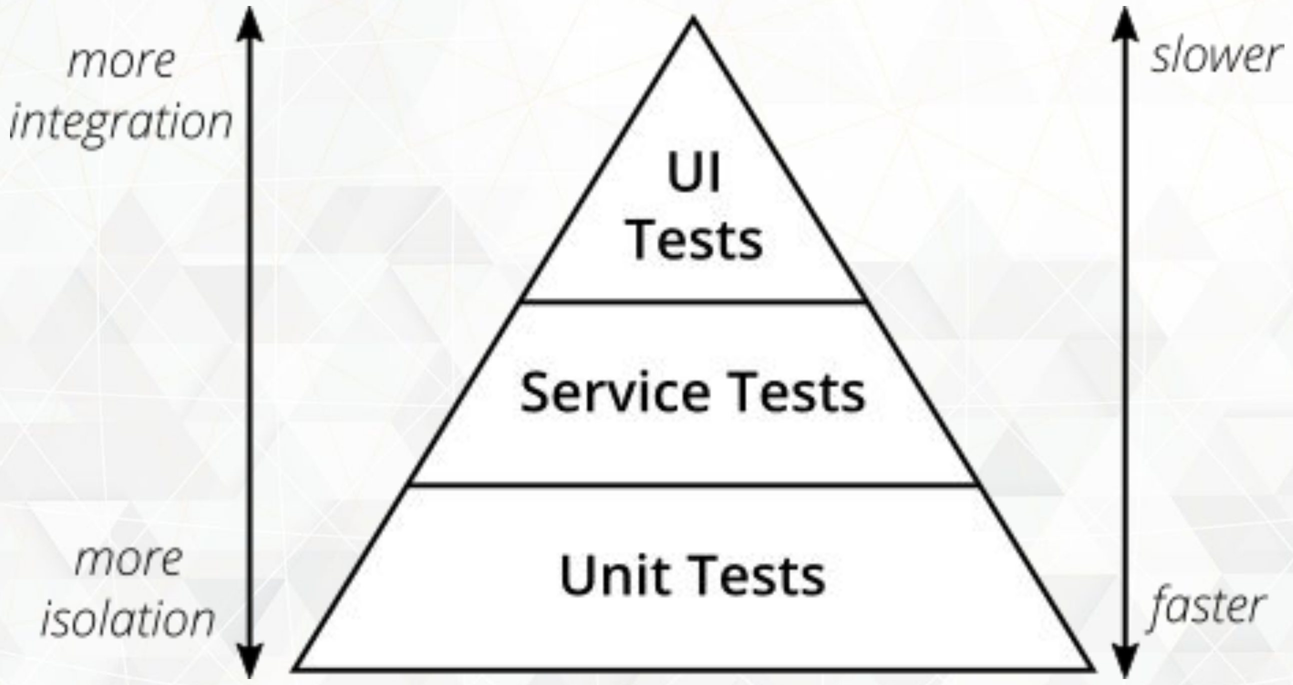
2. Rather than just focusing attention on the input conditions (input space), test cases are also derived by considering the result space (output equivalence classes).

## Boundary Value Analysis

- R3. The player is entitled to two shots to reach the maximum score (10) on each round.

| Test Case | Result |
|---|---|
| Score (-1) | Invalid value |
| Score (11) | Invalid value |
| Score (0) | Score saved |
| Score (10) | Score saved |

## The Test Pyramid



## The Test Pyramid

Mike Cohn's original test pyramid consists of three layers that your test suite should consist of (bottom to top):

1. Unit Tests
2. Service Tests
3. User Interface Tests

## Test Doubles

Meszaros uses the term Test Double as the generic term for any kind of pretend object used in place of a real object for testing purposes.

- Dummy
- Fake
- Stubs
- Spies
- Mocks

## Dummy

Objects are passed around but never actually used. Usually they are just used to fill parameter lists.

## Dummy

```java
@Test
public void shouldStandardize() {
    WidgetDaoDummy widgetDummy = new WidgetDaoDummy();
    WidgetService widgetService = new WidgetService(widgetDummy);
    widgetService.standardize();
    assertTrue(widget.isStandardized());
}
class WidgetDaoDummy implements WidgetDao {
   @Override
   public Widget getWidget() {
        throw new RuntimeException("Not expected to be called");
   }
   @Override
   public void saveWidget(Widget widget) {
        throw new RuntimeException("Not expected to be called");
   }
}
```

## Fake

Objects actually have working implementations, but usually take some shortcut which makes them not suitable for production (an in memory database is a good example).

## Fake

```java
public void shouldListFlights() throws Exception{
    FlightMgmtFacadeImpl facade = new FlightMgmtFacadeImpl();
    facade.setDao(new InMemoryDatabase());
    BigDecimal yyc = facade.createAirport("YYC", "Calgary", "Calgary");
    BigDecimal lax = facade.createAirport("LAX", "LAX Intl", "LA");
    facade.createFlight(yyc, lax);
    List flights = facade.getFlightsByOriginAirport(yyc);
    assertEquals( "# of flights", 1, flights.size());
    Flight flight = (Flight) flights.get(0);
    assertEquals( "origin", yyc, flight.getOrigin().getCode());
 }
```

## Fake

```
public class InMemoryDatabase implements FlightDao {
   private List airports = new Vector();
   public Airport createAirport(String airportCode, String name, String nearbyCity)
            throws DataException, InvalidArgumentException {
      Airport result = new Airport(getNextAirportId(), airportCode, name, nearbyCity);
      airports.add(result);
      return result;
   }
   public Airport getAirportByPrimaryKey(BigDecimal airportId) {
      Iterator i = airports.iterator();
      while (i.hasNext()) {
         Airport airport = (Airport) i.next();
         if (airport.getId().equals(airportId)) {
            return airport;
         }
      }
      throw new DataException("Airport not found:"+airportId);
}
```

## Stubs

Provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test.

## Stubs

```java
public void shouldDisplayCurrentTimeAtMidnight() throws Exception {
      TimeProvider timeProviderStub = new TimeProviderStub();
      TimeDisplay timeDisplay = new TimeDisplay();
      timeDisplay.setTimeProvider(timeProviderStub);
      String result = timeDisplay.getCurrentTimeAsHtmlFragment();
      String expected = "<span>Midnight</span>";
      assertEquals("Display Current Time Midnight", expected, result);
}

public class TimeProviderStub extends TimeProvider {
   public Calendar getTime() {
      Calendar myTime = new GregorianCalendar();
      myTime.set(Calendar.HOUR_OF_DAY, hours);
      myTime.set(Calendar.MINUTE, minutes);
      return myTime;
   }
}
```

## Spies

Are stubs that also record some information based on how they were called. One form of this might be an email service that records how many messages it was sent.

## Spies

```
public void shouldAuditRemoveFlight() throws Exception {
      FlightManagementFacade facade = new FlightManagementFacadeImpl();
      AuditLog auditLogSpy = new AuditLogSpy()
      facade.setAuditLog(auditLogSpy);
      facade.removeFlight(13);
      assertEquals("detail", 13, auditLogSpy.getDetail());
}
public class AuditLogSpy implements AuditLog {
   private Object detail;
   public void logMessage(Object detail) {
      this.detail = detail;
   }
   public Object getDetail() {
      return detail;
   }
}
```

## Mocks

Are what we are talking about here: objects pre-programmed with expectations which form a specification of the calls they are expected to receive.

## Mocks

```java
public void shouldRemoveFlight() throws Exception {
      FlightDto expectedFlightDto = createAnonRegFlight();
      ConfigurableMockAuditLog mock = new ConfigurableMockAuditLog();
      mock.setExpectedLogMessage(expectedFlightDto.getFlightNumber());
      FlightManagementFacade facade = new FlightManagementFacadeImpl();
      facade.setAuditLog(mock);
      facade.removeFlight(expectedFlightDto.getFlightNumber());
      Boolean exists = facade.flightExists(expectedFlightDto.getFlightNumber());
      assertFalse("flight still exists after being removed", exists);
      mock.verify();
   }
```

## Unit Tests

A level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.

## Integration Tests

A level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

## E2E Tests

A level of software testing where a complete and integrated software is tested. The purpose of this test is to evaluate the system’s compliance with the specified requirements.

## Strategy Development

- Test Addition
- Test Last
- Test First
- Refactoring
- Test Driven Development

## Actions

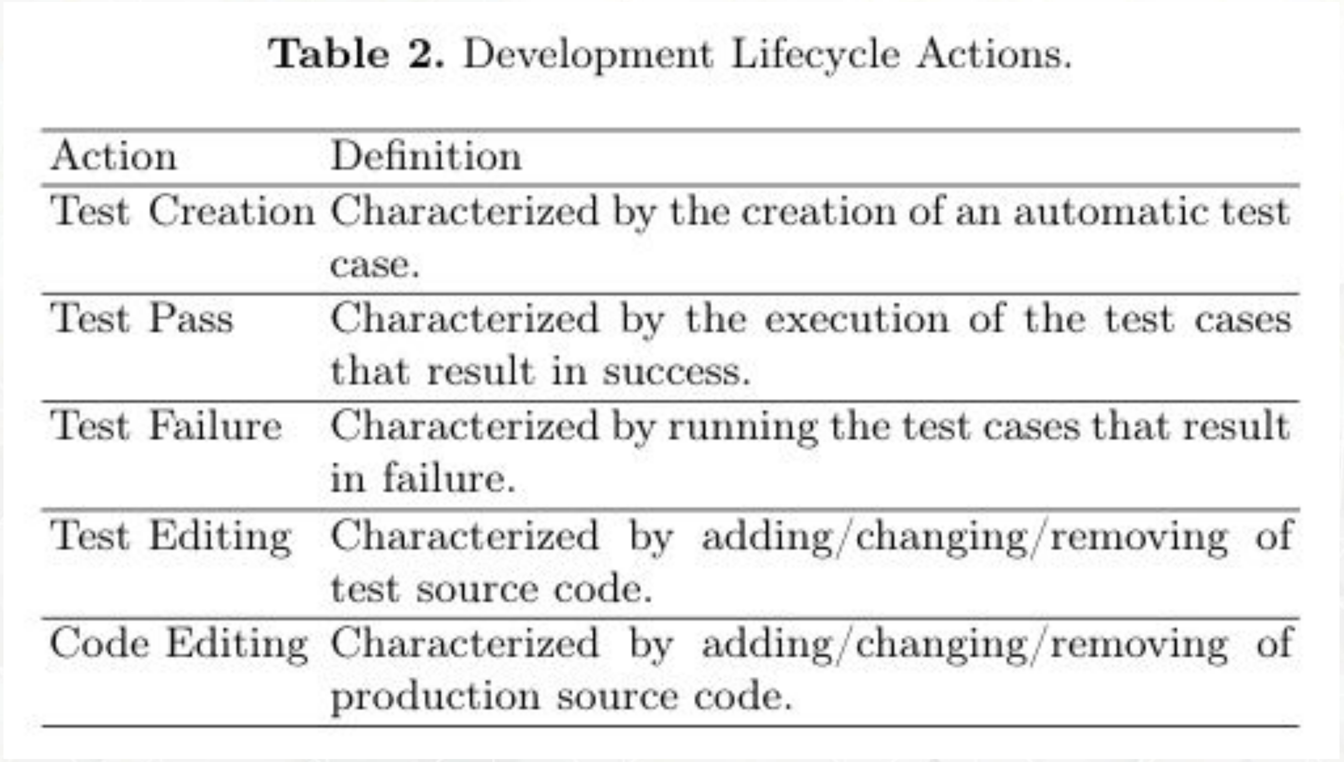
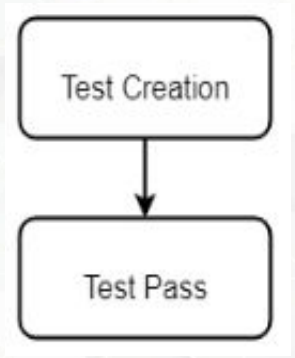
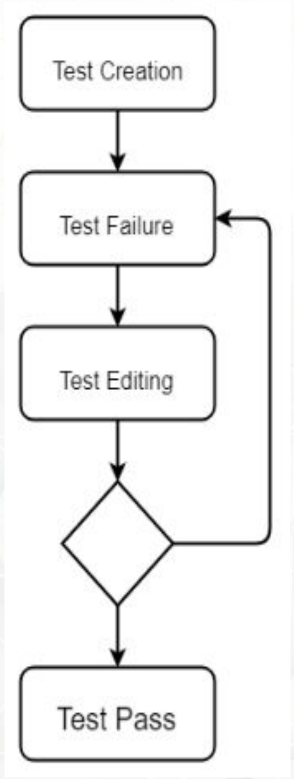


**Table 2.** Development Lifecycle Actions.

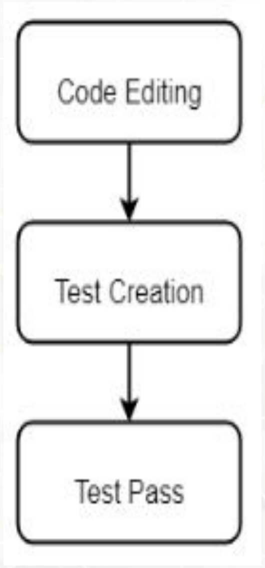
| Action | Definition |
|---|---|
| Test Creation | Characterized by the creation of an automatic test case. |
| Test Pass | Characterized by the execution of the test cases that result in success. |
| Test Failure | Characterized by running the test cases that result in failure. |
| Test Editing | Characterized by adding/changing/removing of test source code. |
| Code Editing | Characterized by adding/changing/removing of production source code. |

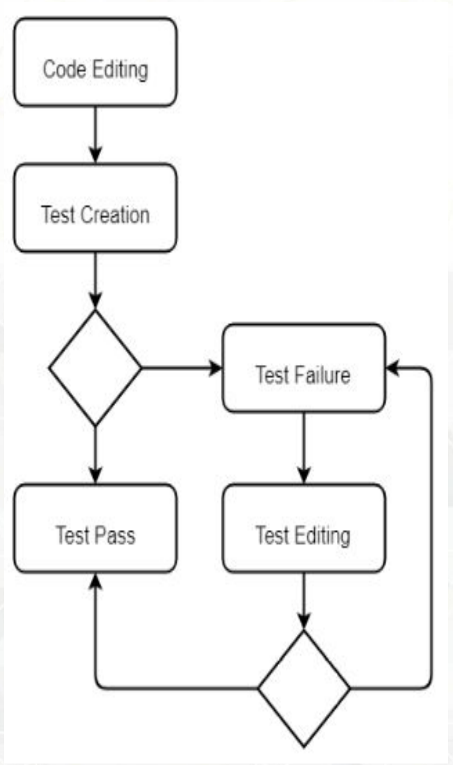
# Test Addition



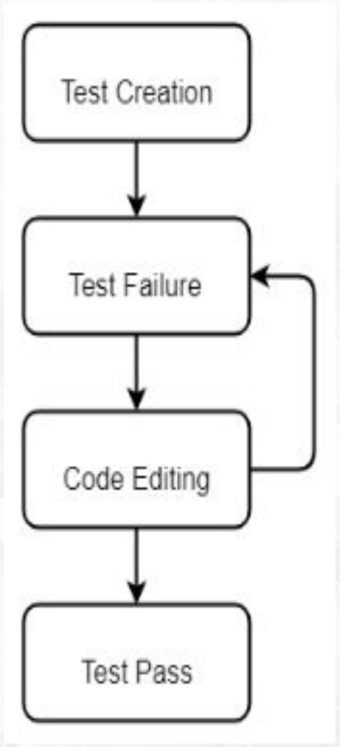


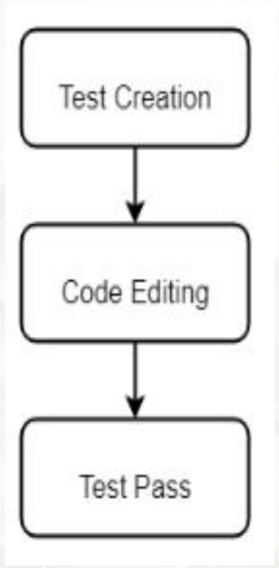
## Test Last

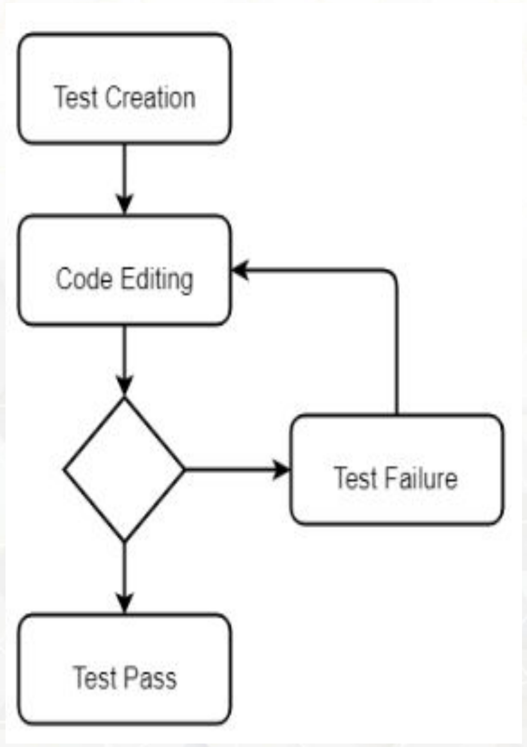

Code Editing
Test Creation
Test Pass


Code Editing
Test Creation
Test Failure
Test Pass
Test Editing

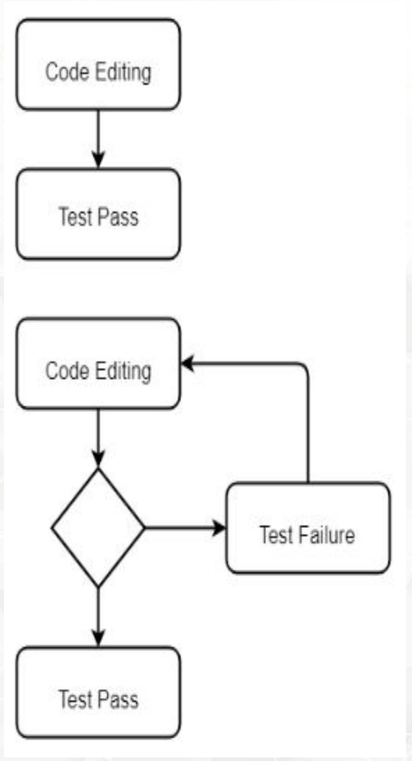
## Test First

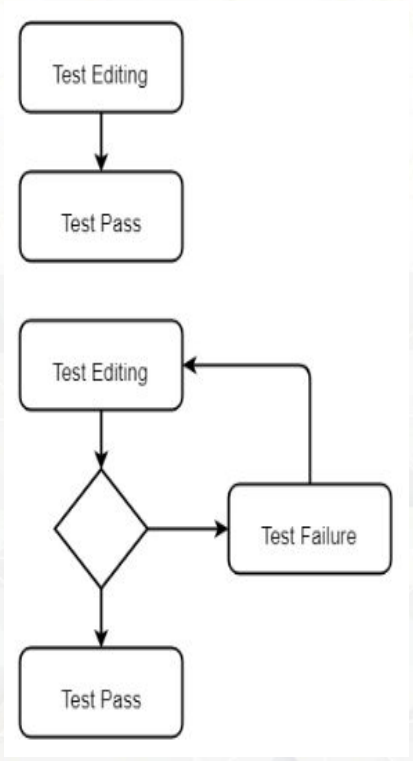

Test Creation
Test Failure
Code Editing
Test Pass


Test Creation
Code Editing
Test Pass

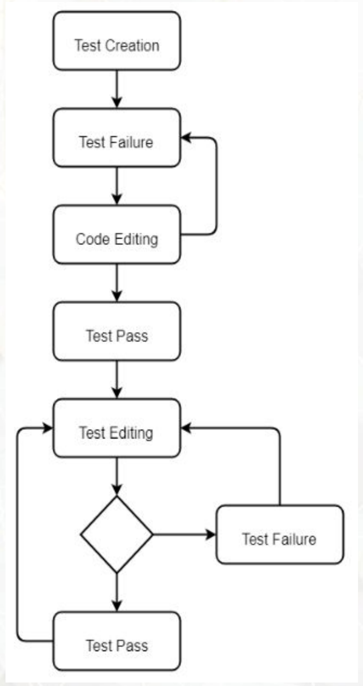

Test Creation
Code Editing
Test Failure
Test Pass

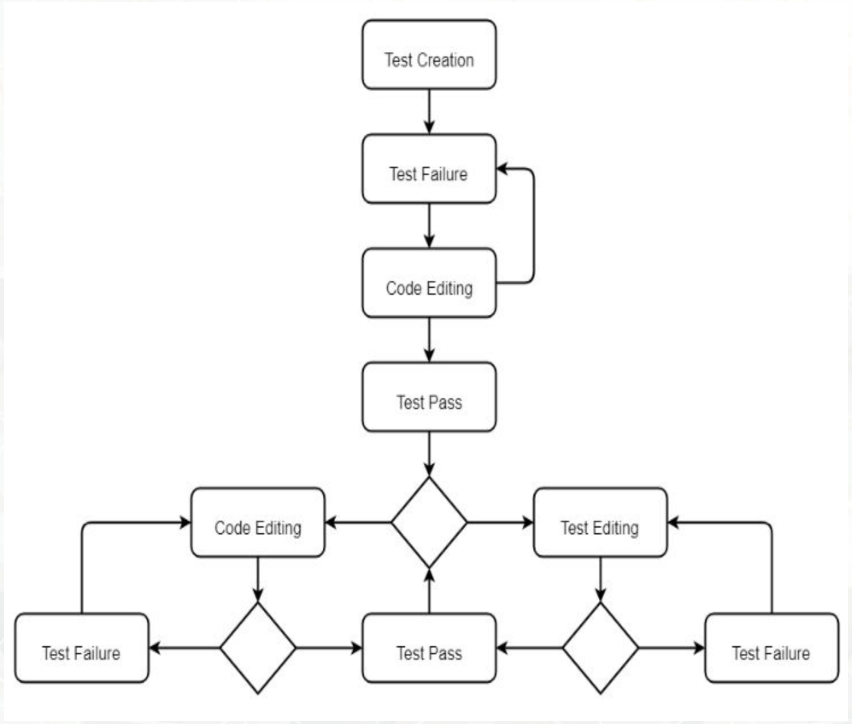
# Refactoring

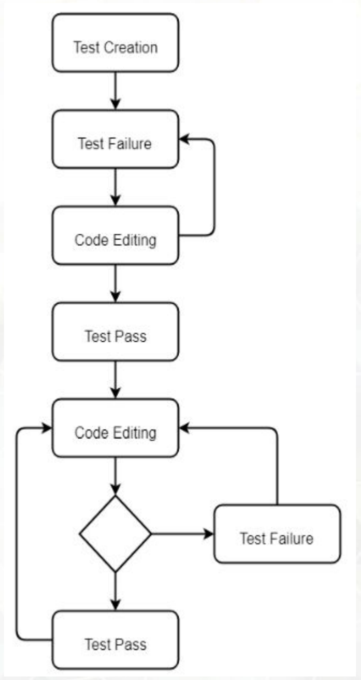




# Test Driven Development



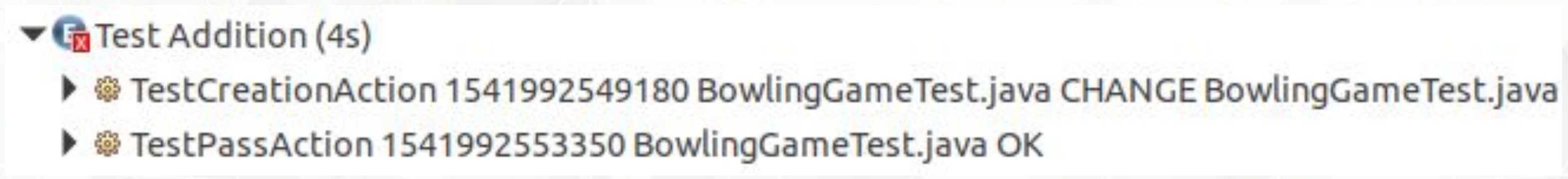




## Butterfly

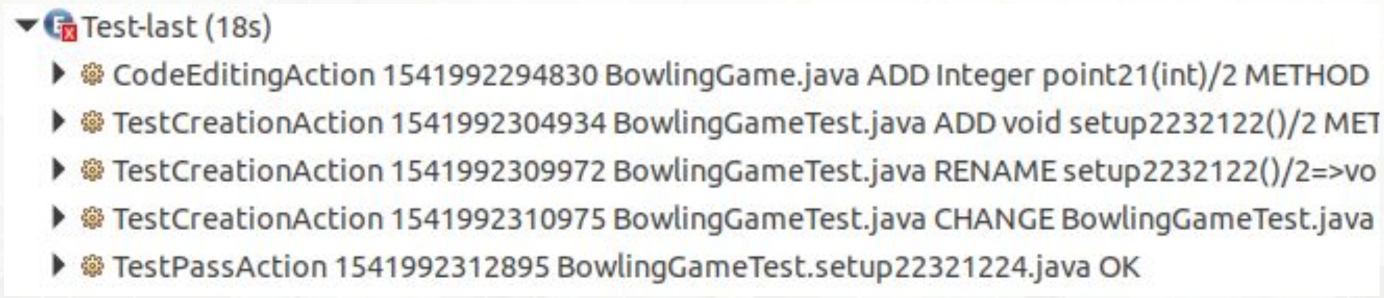
- An plugin for Eclipse IDE
- Based in user actions
- Open-source
- Transparent for users
- Monitoring Total Lifecycle of TDD

## Test Addition



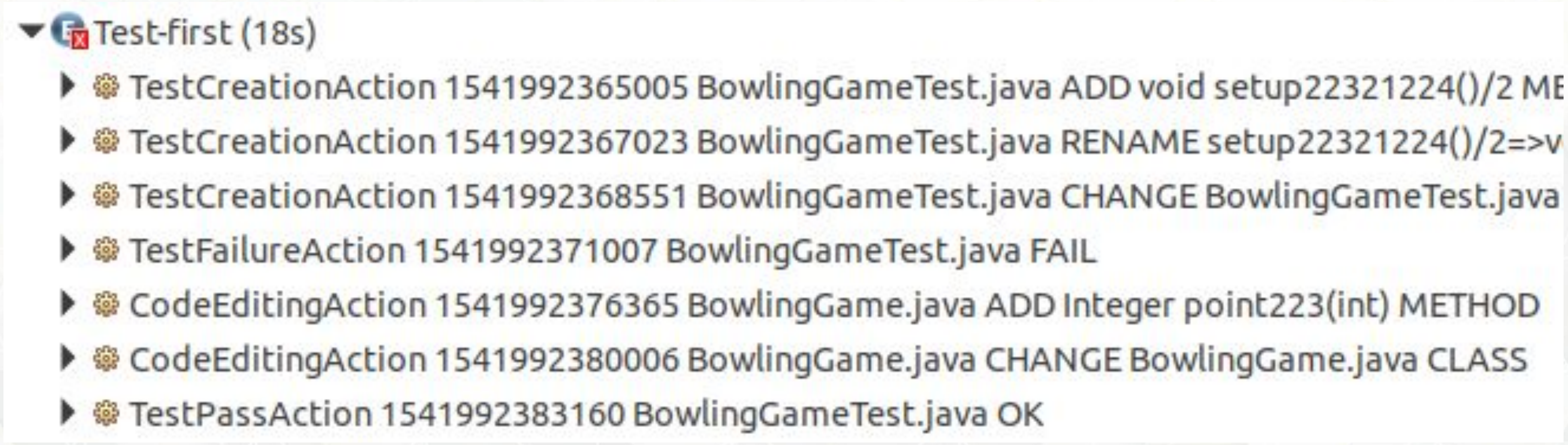
```
▾ Test Addition (4s)
    ▸ TestCreationAction 1541992549180 BowlingGameTest.java CHANGE BowlingGameTest.java
    ▸ TestPassAction 1541992553350 BowlingGameTest.java OK
```

## Test Last



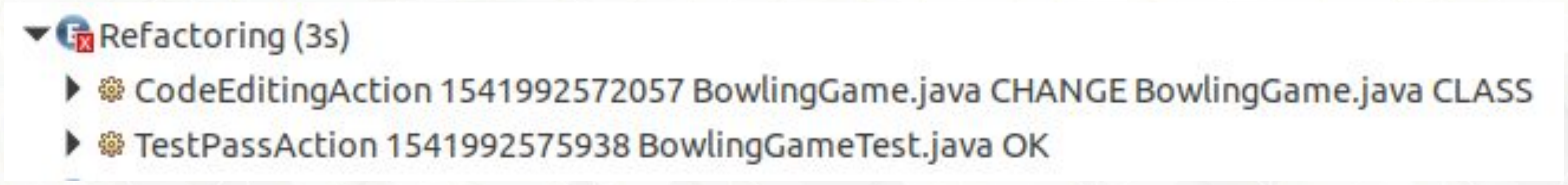
```
▼ Test-last (18s)
  ▶ CodeEditingAction 1541992294830 BowlingGame.java ADD Integer point21(int)/2 METHOD
  ▶ TestCreationAction 1541992304934 BowlingGameTest.java ADD void setup2232122()/2 MET
  ▶ TestCreationAction 1541992309972 BowlingGameTest.java RENAME setup2232122()/2=>vo
  ▶ TestCreationAction 1541992310975 BowlingGameTest.java CHANGE BowlingGameTest.java
  ▶ TestPassAction 1541992312895 BowlingGameTest.setup22321224.java OK
```

## Test First



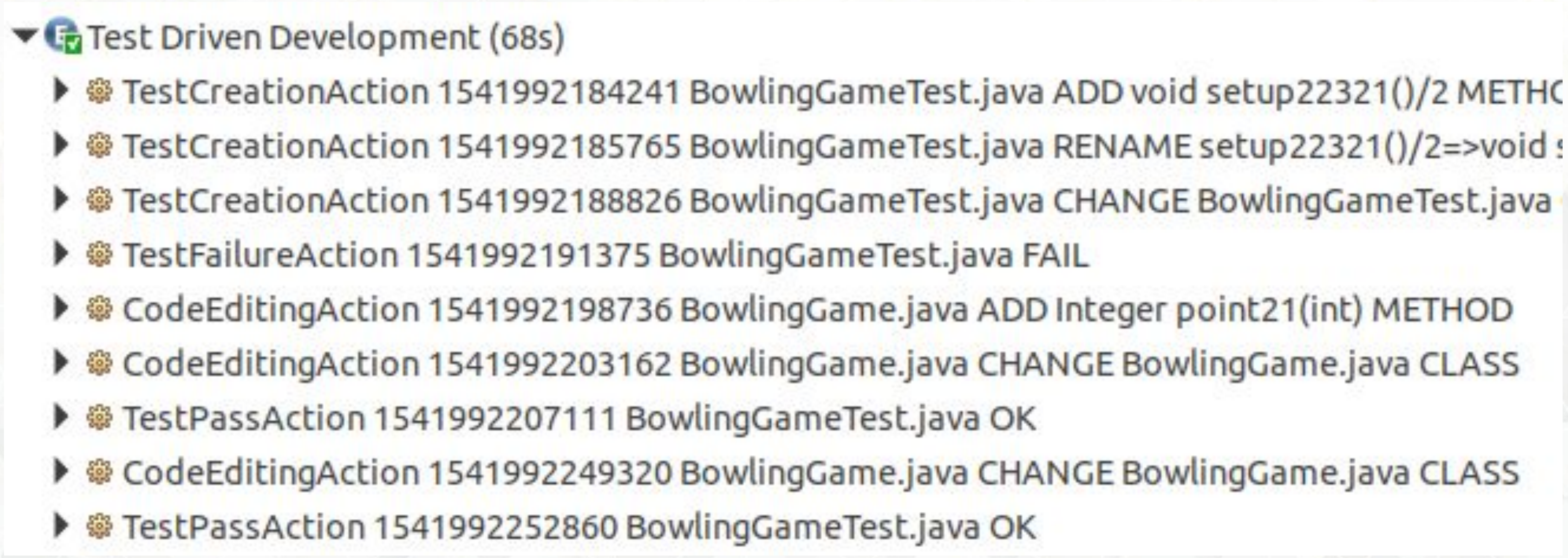
```
Test-first (18s)
  TestCreationAction 1541992365005 BowlingGameTest.java ADD void setup22321224()/2 ME
  TestCreationAction 1541992367023 BowlingGameTest.java RENAME setup22321224()/2=>v
  TestCreationAction 1541992368551 BowlingGameTest.java CHANGE BowlingGameTest.java
  TestFailureAction 1541992371007 BowlingGameTest.java FAIL
  CodeEditingAction 1541992376365 BowlingGame.java ADD Integer point223(int) METHOD
  CodeEditingAction 1541992380006 BowlingGame.java CHANGE BowlingGame.java CLASS
  TestPassAction 1541992383160 BowlingGameTest.java OK
```

## Refactoring



```
Refactoring (3s)
  CodeEditingAction 1541992572057 BowlingGame.java CHANGE BowlingGame.java CLASS
  TestPassAction 1541992575938 BowlingGameTest.java OK
```

# Test Driven Development



```
Test Driven Development (68s)
  TestCreationAction 1541992184241 BowlingGameTest.java ADD void setup22321()/2 METHC
  TestCreationAction 1541992185765 BowlingGameTest.java RENAME setup22321()/2=>void
  TestCreationAction 1541992188826 BowlingGameTest.java CHANGE BowlingGameTest.java
  TestFailureAction 1541992191375 BowlingGameTest.java FAIL
  CodeEditingAction 1541992198736 BowlingGame.java ADD Integer point21(int) METHOD
  CodeEditingAction 1541992203162 BowlingGame.java CHANGE BowlingGame.java CLASS
  TestPassAction 1541992207111 BowlingGameTest.java OK
  CodeEditingAction 1541992249320 BowlingGame.java CHANGE BowlingGame.java CLASS
  TestPassAction 1541992252860 BowlingGameTest.java OK
```

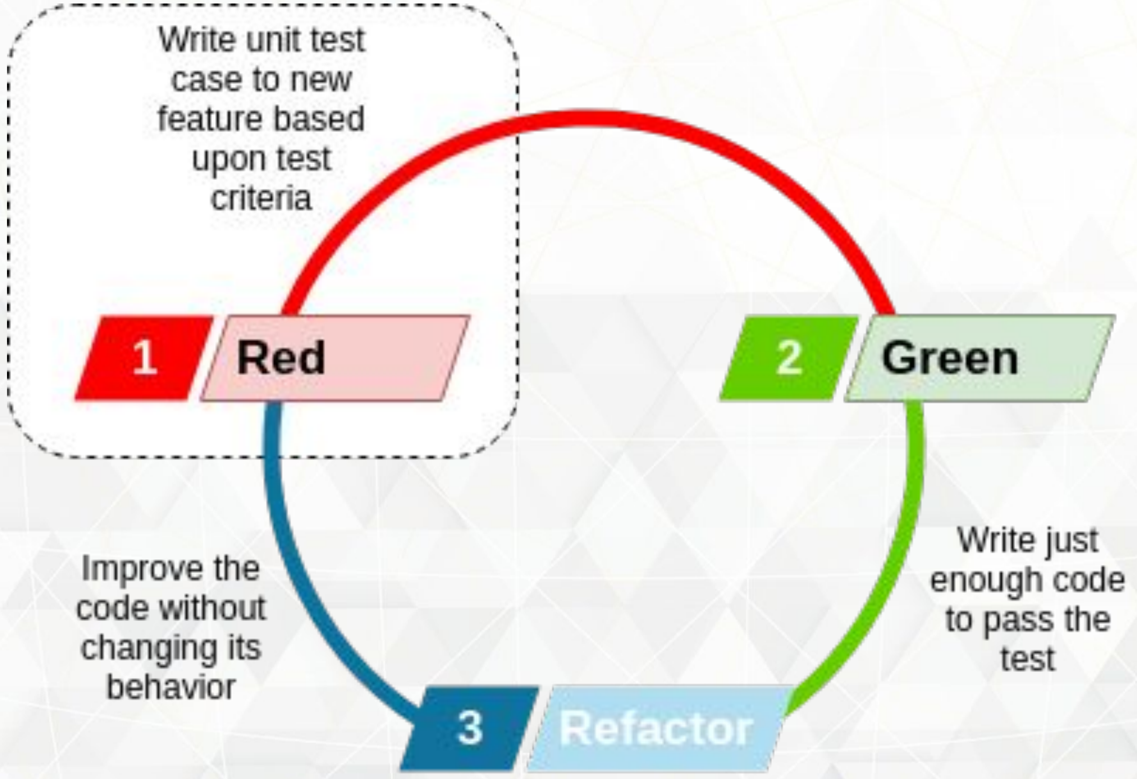
## Bowling Game Requirements

- R1. The score of the game can be consulted at any time.
- R2. The player is entitled to two shots to reach the maximum score (10) on each round.
- R3. The game consists of 10 rounds.
- R4. If on the first shot the maximum score is reached (strike), the player will not be entitled to the second shoot.

## Bowling Game Requirements

- R5. If in both shot the maximum score is reached (spare), the player will have as bonus the score obtained in the next move.
- R6. The strike score bonus is the value of the next two moves.
- R7. The player will have two extra shots if he strikes in the tenth round.
- R8. If he reaches the spare in the two shots after the tenth round, the player will be entitled to one extra shot.

## Steps


Write unit test case to new feature based upon test criteria
1 Red
2 Green
Write just enough code to pass the test
3 Refactor
Improve the code without changing its behavior

## Bowling Game Training

## Bowling Game Microservice

# Software Testing Training

Aluno: Altieres de Matos
altitdb@gmail.com

Orientador: Prof. Dr. Marco Aurélio Graciotto Silva
Coorientador: Prof. Dr. Reginaldo Ré