Test Automation

Idaho State University | Computer Science

Isaac Griffith

CS 4422 and CS 5599
Department of Computer Science
Idaho State University

ROAR

Outcomes

At the end of Today's Lecture you will be able to:

- Understand the basic concepts of test automation
- Understand Testability, Observability, and Controllability
- Understand and be capable of using JUnit

Inspiration

“The principle objective of software testing is to give confidence in the software.” – Anonymous

What is Test Automation?

Using software to control the testing

- **Setting up** test preconditions
- Test **execution**
- **Comparing** actual results to test results

- Reduces **cost**
- Reduces **human error**
- Reduces **variance** in test quality from different individuals
- Significantly reduces the cost of **regression** testing

Software Testability

Testability

The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.

- **How hard is it to find faults** in the software
- Testability is dominated by **two** practical problems
  - How to **observe the results** of test execution
  - How to **provide the test values** to the software

Idaho State University
Computer Science

Observability and Controllability

Observability

How easy it is to observe the behavior of a program in terms of its outputs, effects on the environment and other hardware and software components.

- Software that affects hardware devices, databases or remote files have low observability

Controllability

How easy it is to provide a program with the needed inputs, in terms of values, operations, and behaviors.

- Easy to control software with inputs from keyboards
- Inputs from hardware sensors or distributed software is harder.
- **Data abstraction** reduces controllability and observability

ROAR

Components of a Test Case

- A test case is a **multipart artifact** with a definite structure
  - Test Case Values: **The input values needed to complete an execution of the software under test**
  - Expected Results: **The result that will be produced by the test if the software behaves as expected.**
    - A **test oracle** uses expected results to decide whether a test passed or failed.

Controllability and Observability

Aspects affecting Controllability and Observability

- Prefix values
  - **Inputs to put the software into the correct state to receive the test case values**
- Postfix values
  - **Inputs that must be sent to the software after the test case values**

Putting Tests Together

- Test Case:
  - **The test case values, prefix values, postfix values, and expected results necessary for a complete execution and evaluation of the software under test.**
- Test set (or suite)
  - **A set of test cases**
- Executable test script:
  - **A test case that is prepared in a form to be executed automatically on the test software and produce a report**

Test Automation Framework

A set of assumptions, concepts, and tools that support test automation

JUnit

JUnit Test Framework

- JUnit can be used **to test** …
  - … an entire object
  - … part of an object – a method or some interacting methods
  - … interaction between several objects
- It is primarily intended for unit and integration testing, not system testing
- Each test is embedded into one **test method**
- A **test class** contains one or more test methods
- Test classes **include**:
  - A collection of **test methods**
    - Methods to **set up** the state before and **update** the state after each test and before and after all tests
- Get started at **junit.org**

JUnit Test Fixtures

- A **test fixture** is the **state** of the test
  - Objects and variables that are used by more than one test
  - Initializations (prefix values)
  - Reset values (postfix values)
- Different tests can **use** the objects without sharing the state
- Objects used in test fixtures should be declared as **instance variables**
- They should be initialized in a **@Before** method
- Can be deallocated or reset in an **@After** method

Simple JUnit Example

```
public class Calc {
  public static int add(int a, int b) {
    return a + b;
  }
}

import org.junit.Test;
import static org.junit.Assert.*;

public class CalcTest {
  @Test public void testAdd() {
    assertEquals("testAdd incorrect", 5,
    Calc.add(2, 3));
  }
}
```

- `testAdd incorrect` printed if assert fails
- Expected Value: `5`
- Test Values: `[2, 3]`

Testing the Min Class

```
import java.util.*;

public class Min
{
  /**
    * Returns the minimum element in a list
    * @param list Comparable list of elements to search
    * @return the minimum element in the list
    * @throws NullPointerException if list is null or
    *         if any list elements are null
    * @throws ClassCastException if list elements are not mutually comp
    * @throws IllegalArgumentException if list is empty
    */
     …
}
```

Testing the Min Class

```java
public static <T extends Comparable<? super T>> T min
(List<? extends T> list) {
  if (list.size() == 0) {
    throw new IllegalArgumentException("Min.min");
  }
  Iterator<? extends T> itr = list.iterator();
  T result = itr.next();

  if (result == null) throw new NullPointerException
  ("Min.min");

  while (itr.hasNext()) {
    // throws NPE, CCE as needed
    T comp = itr.next();
    if (comp.compareTo(result) < 0) {
      result = comp;
    }
  }
```

In-Class Exercise

Individual Exercise:

1. Write test inputs for the Min class
2. Be sure to include expected outputs
3. Once you have enough tests, write one in JUnit
4. If you’re not sure how, ask for help
5. If you have written JUnit tests, help someone who has not
6. You do not need to execute the tests.

MinTest Class

```java
import static org.junit.Assert.*;
import org.junit.*;
import java.util.*;

private List<String> list; //Test fixture

// Set up - Called before every test method
@Before // prefix
public void setUp() {
  list = new ArrayList<String>();
}

// Tear down - Called after every test method
@After // postfix
public void tearDown() {
  list = null; // redundant in this example
}
```

Min Test Cases: NullPointerException

```
@Test
public void testForNullList()
{
  list = null;
  try {
    Min.min(list);
  } catch (NullPointerException e)
  {
    return;
  }
  fail("NullPointerException"+
    "expected");
}
```

This NPE test uses the **fail** assertion

This NPE test catches an easily overlooked special case. ->

This NPE test decorates the **@Test** annotation with the class of the exception

```
@Test(expected =
NullPointerException.class)
public void testForNullElement()
{
  list.add(null);
  list.add("cat");
  Min.min(list);
}
```

```
@Test(expected =
NullPointerException.class)
public void testForSoloNullElement()
{
  list.add(null);
  Min.min(list);
}
```

More Exception Test Cases for Min

Note that Java generics don't prevent clients from using raw types!

```
@Test(expected =
ClassCastException.class)
@SuppressWarnings("unchecked")
public void
testMutuallyIncomparable()
{
   List list = new ArrayList();
   list.add("cat");
   list.add("dog");
   list.add(1);
   Min.min(list);
}
```

```
@Test(expected =
IllegalArgumentException.class)
public void testEmptyList()
{
      Min.min(list);
}
```

Special case: Testing for the empty list

Remaining Test Cases for Min

- Finally! A couple of “Happy Path” tests

```
@Test
public void testSingleElement()
{
  list.add("cat")
  Object obj = Min.min(list);
  assertTrue("Single Element List",
  obj.equals("cat"));
}
```

```
@Test
public void testDoubleElement()
{
  list.add("dog");
  list.add("cat");
  Object obj = Min.min(list);
  assertTrue("Double Element list",
  obj.equals("cat"));
}
```

Summary: Seven Tests for Min

- Five tests with exceptions
  - ❶ null list
  - ❷ null element with multiple elements
  - ❸ null single element
  - ❹ incomparable types
  - ❺ empty elements
- Two without exceptions
  - ❻ single element
  - ❼ two elements

JUnit Resources

- Some JUnit tutorials
- JUnit: Download, Documentation

Summary

- The only way to make testing **efficient** as well as **effective** is to **automate** as much as possible
- Test frameworks provide very simple ways to **automate** our tests
- It is no “**silver bullet**” however … it does not solve the hard problem of testing:
- This is test design … the purpose of **test criteria**

Are there any questions?