Introduction to Software Testing
Test Automation (Ch. 3)
Testing activities

Testing can be broken up into four general types of activities

1. **Test design**
   - 1.a. **Criteria based**
   - 1.b. **Human-based**

2. **Test automation**
3. **Test execution**
4. **Test evaluation**
What is test automation?

Using software to control the testing

- **Setting up** test preconditions
- Test **execution**
- **Comparing** actual results to test results
- Test **reporting**
What is test automation?

Using software to control the testing
- **Setting up** test preconditions
- **Test execution**
- **Comparing** actual results to test results
- **Test reporting**

Reduces **cost**
Reduces **human error**
Reduces **variance** in test quality from different individuals

Significantly reduces the cost of **regression** testing

![TEST ALL THE THINGS? AUTOMATE ALL THE THINGS!](image)
Software testability (3.1)

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.
Software testability (3.1)

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 test values to the software
Observability & 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.
Observability & Controllability

**Observability**

*How easy it is to observe the behavior of a program in term 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
Observability & Controllability

Observability
How easy it is to observe the behavior of a program in term 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
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
Components of a test case (3.2)

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
Affecting controllability & observability

Prefix values

*Inputs to put the software into the correct state to receive the test case values*
Affecting controllability & 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 it all 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 (3.3)

A set of assumptions, concepts, and tools that support test automation.
JUnit for Java test automation

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 systems testing.
JUnit for Java test automation

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 systems 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)
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](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/invoke/AnnotationTarget.html) method

Can be deallocated or reset in an [@After](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/invoke/AnnotationTarget.html) method
public class Calc {
    static public 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() {
        assertTrue("testAdd incorrect", 5 == Calc.add(2, 3));
    }
}
public class Calc {
    static public 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() {
        assertTrue("testAdd incorrect",
                    5 == Calc.add(2, 3));
    }
}
Simple JUnit Example

```java
public class Calc {
    static public 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() {
        assertTrue("testAdd incorrect", 5 == Calc.add(2, 3));
    }
}
```

Expected output Printed if assert fails
public class Calc {
    static public 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() {
        assertTrue("testAdd incorrect", 5 == Calc.add(2, 3));
    }
}
Testing the Min class

```java
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 comparable
     * @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;
        }
    }
    return result;
}
```
In-class Exercise

Write **test inputs for the Min class**

Be sure to **include expected outputs**

Once you have enough tests, **write one in JUnit**.

If you’re not sure how, **ask for help**.
If you have written JUnit tests, **help somebody who has not**.

**You do not need to execute the tests.**
MinTest class

Standard imports for all JUnit classes:

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

Test fixture and pre-test setup method (prefix):

```java
private List<String> list;   // Test fixture

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

Post test teardown method (postfix):

```java
// Tear down - Called after every test method.
@After
public void tearDown()
{
    list = null;   // redundant in this example
}
Min test cases: NullPointerException

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

@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);
}
```

This NullPointerException test
uses the fail assertion

This NullPointerException test
decorates the @Test annotation with
the class of the exception

This NullPointerException test
catches an easily overlooked
special case
More exception tests for Min

```java
@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);
}
```

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

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

Special case: Testing for the empty list
Remaining tests for Min

```java
@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"));
}
```

Finally! A couple of “Happy Path” tests
Summary: Seven tests for Min

Five tests for exceptions
1. null list
2. null element with multiple elements
3. null single element
4. incomparable types
5. empty elements

Two without exceptions
1. single element
2. two elements
JUnit Resources

Some JUnit tutorials
- https://www.tutorialspoint.com/junit/index.htm
- https://www.vogella.com/tutorials/JUnit/article.html

JUnit: download and documentation
- http://www.junit.org
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:

*What test values to use?*

This is test design – the purpose of *test criteria*.