Test Automation

Software Testing

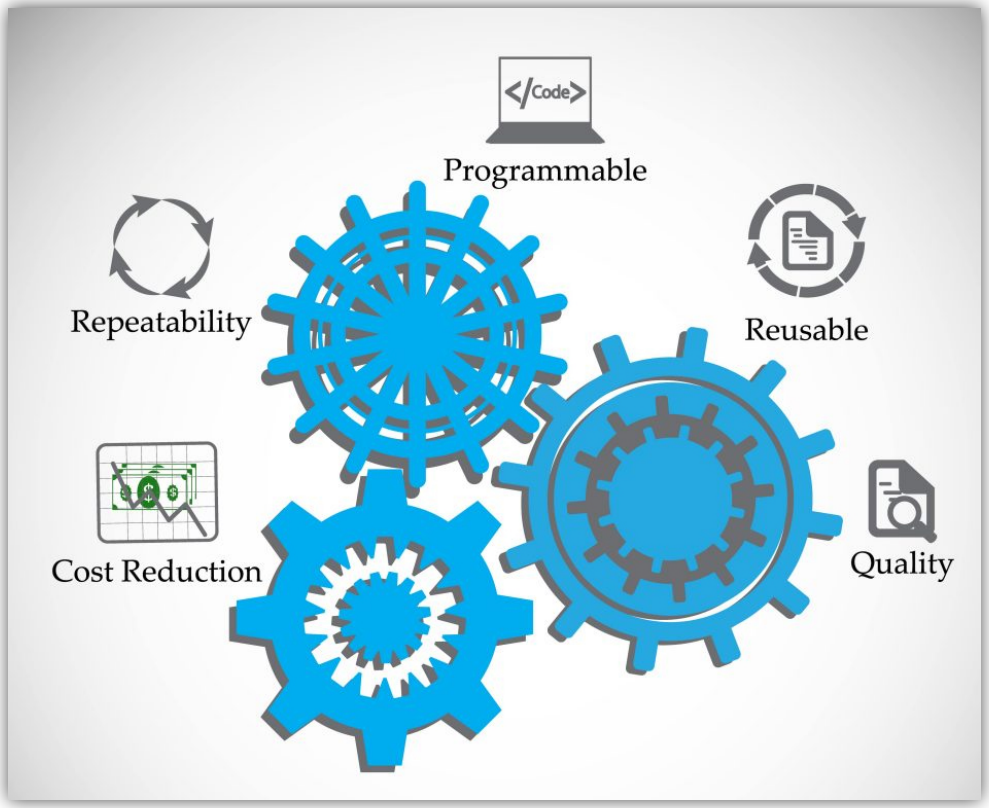
(3104313)

Amirkabir University of Technology

Spring 1399-1400

Test Automation

The use of software to control the **execution** of tests, the **comparison** of actual outcomes to predicted outcomes, the **setting up** of test preconditions, and other test **control** and test **reporting** functions.



Software 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.

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

Controllability

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

Test Case

- Test case values
- Prefix values
- Postfix values
- Expected results

Test Case

A test case **is composed of** the test case values, prefix values, postfix values, and expected results necessary for a complete execution and evaluation of the software under test (SUT).

- Test case values
- Prefix values
- Postfix values
- Expected results

- Test Set
- Test Suite

In-Class Exercise #7

- Explain how the components of a test case are related to RIPR model?
- What are the limitations of test automation? Do we always automate the tests?

- You have 5 minutes
- Do the exercise individually/in groups (of 3)
- Upload your answer in Moodle.

Test Case

A test case is composed of the test case values, prefix values, postfix values, and expected results necessary for a complete execution and evaluation of the software under test (SUT).

- Test case values
- Prefix values
- Postfix values
- Expected results

- Test Set
- Test Suite

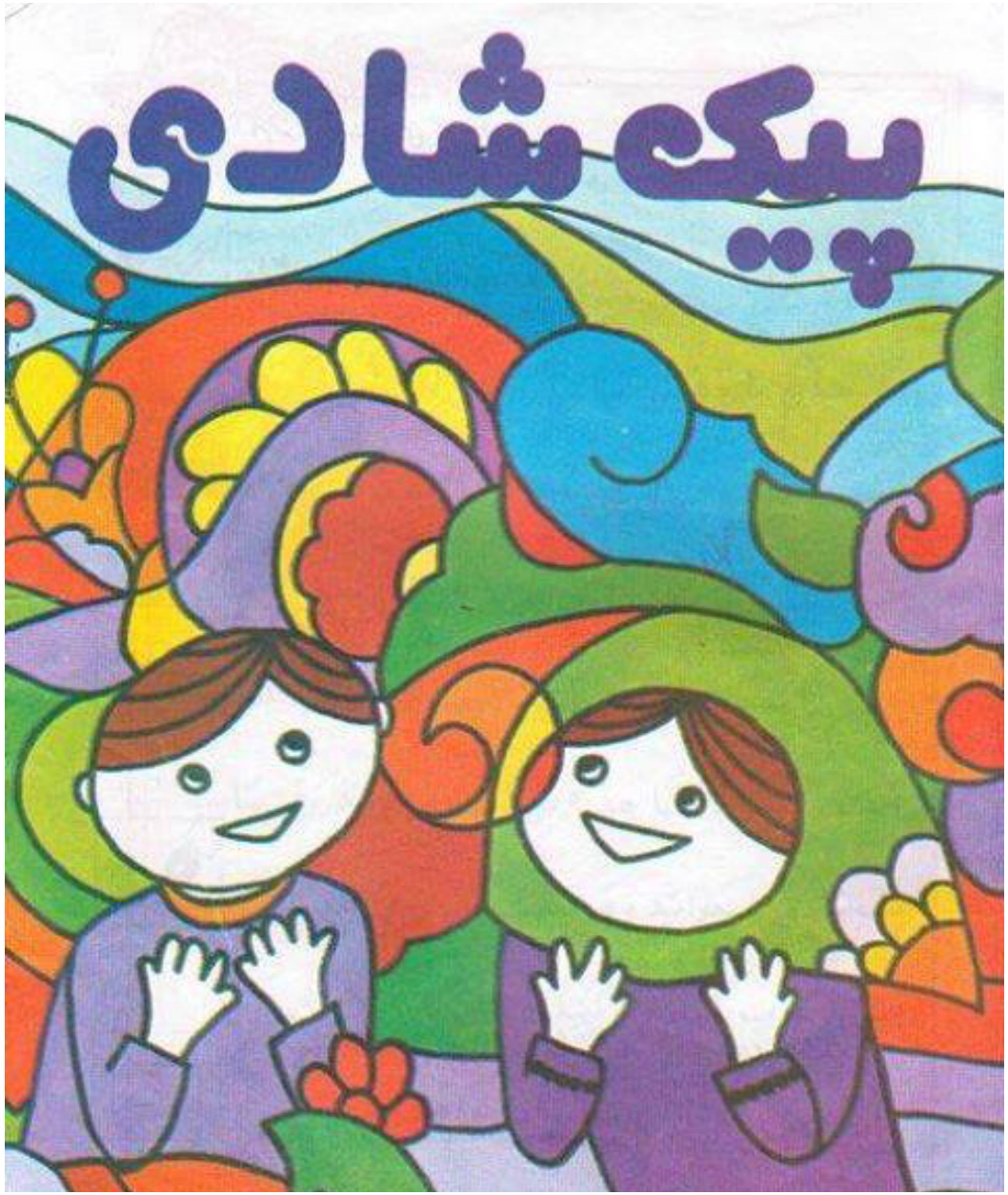
- RIPR Model
  - Reachability
  - Infection
  - Propagation
  - Revealability

Test Automation Framework

A test framework provides a standard **design for test scripts**, and should include support for the **test driver**.

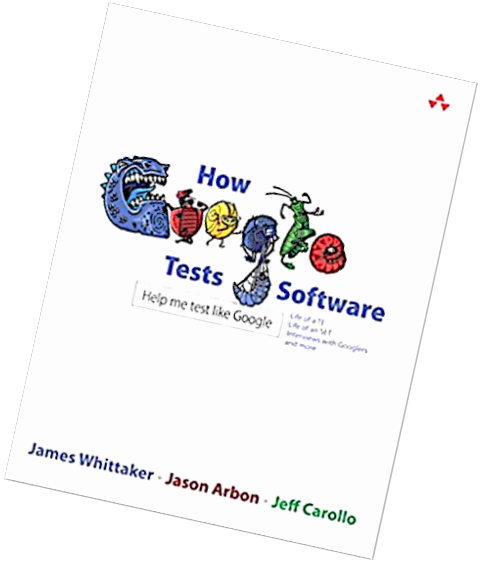
Test Automation Framework

A test framework provides a standard **design for test scripts**, and should include support for the **test driver**.



Recommended readings ☺

* Junit 5
* Property-based Testing
*



Test Automation Framework

A test framework provides a standard **design for test scripts**, and should include support for the **test driver**.

Testing the Min Class

```
import java.util.*;

public class Min
{
  /**
    * Returns the mininum 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

```
imp

pub
{
  /

}
```

```
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 #8

- Write test inputs for the Min class. Be sure to include expected outputs.
  - You can write the tests as name-value pairs with expected outputs Or write them as JUnit tests if you have time or prefer!
  - You do not need to execute the tests

- You have 5-10 minutes
- Do the exercise individually/in groups (of 3)
- Upload your answer in Moodle.

```
@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);
}
```

```
@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);
}
```

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

more tests?

How do you write testing code?

How do you write testing code?

- Standard imports for all JUnit classes
- Test fixture and pre-test setup method (prefix)
- Post test teardown method (postfix)

```
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
 public void setUp()
 {
     list = new ArrayList<String>();
 }
```

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

How do you write testing code?

- Standard imports for all JUnit classes
- Test fixture and pre-test setup method (prefix)
- Post test teardown method (postfix)

- Annotate a method with @org.junit.Test
- Call assertTrue(), assertFalse(), assertNotNull(), assertEqual(), assertSame(), ….

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

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

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

Testing a function **multiple times** with similar values?

Tests with **parameters**?

Testing a function **multiple times** with similar values?

Tests with **parameters**?

- Data-Driven Unit Tests
- JUnit Theories

Testing a function **multiple times** with similar values?

Tests with **parameters**?

- Data-Driven Unit Tests
- JUnit Theories

… more details in the coming assignment!

JUnit Resources

- Some JUnit tutorials
  - http://open.ncsu.edu/se/tutorials/junit/
    (Laurie Williams, Dright Ho, and Sarah Smith)
  - http://www.laliluna.de/eclipse-junit-testing-tutorial.html
    (Sascha Wolski and Sebastian Hennebrueder)
  - http://www.diasparsoftware.com/template.php?content=jUnitStarterGuide
    (Diaspar software)
  - http://www.clarkware.com/articles/JUnitPrimer.html
    (Clarkware consulting)

- JUnit: Download, Documentation
  - http://www.junit.org/

What test values to use?

Test frameworks provide very simple ways to **automate** our tests that make testing **efficient as well as effective.**

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.**