Software testing

JUnit

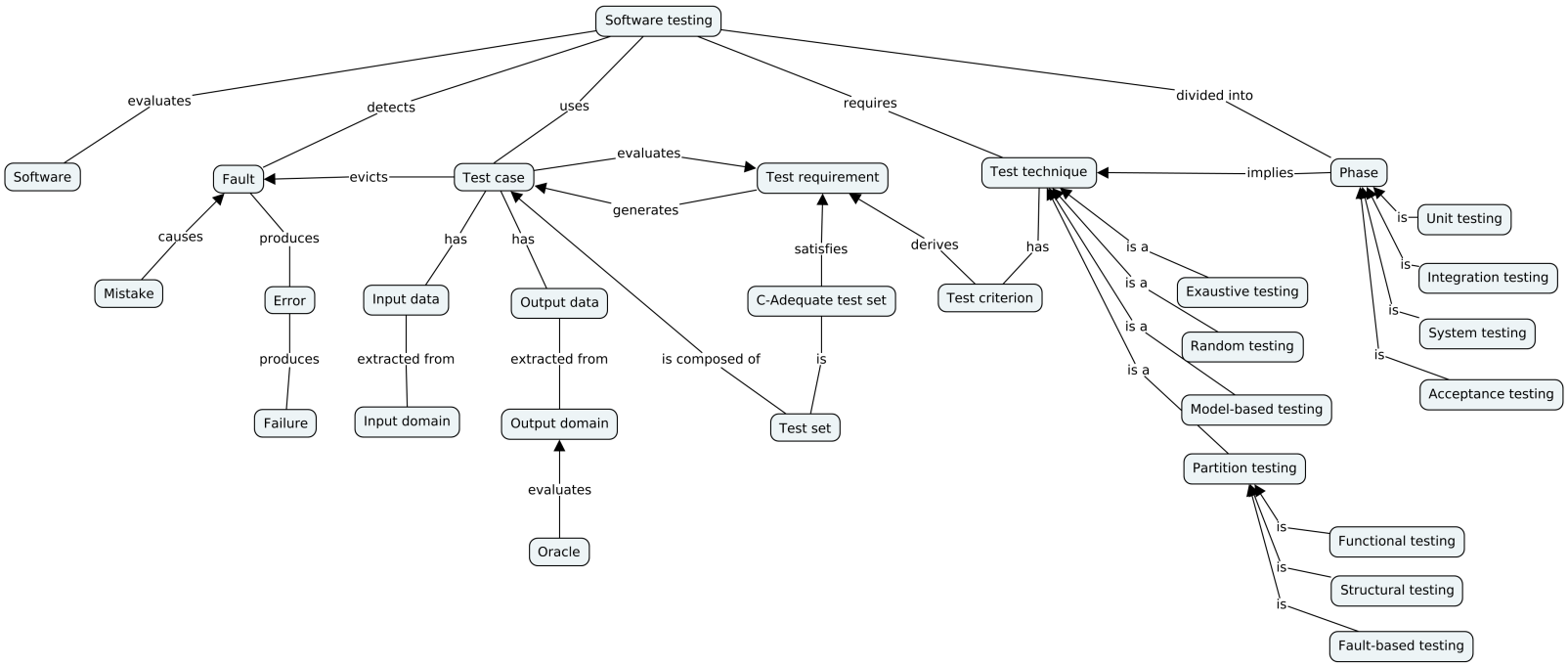
Marco Aurélio Graciotto Silva[1], Ellen Francine Barbosa[2], José Carlos Maldonado[2]

[1]**Department of Computing**
Federal University of Technology – Paraná (UTFPR)
Campo Mourão, PR, Brazil

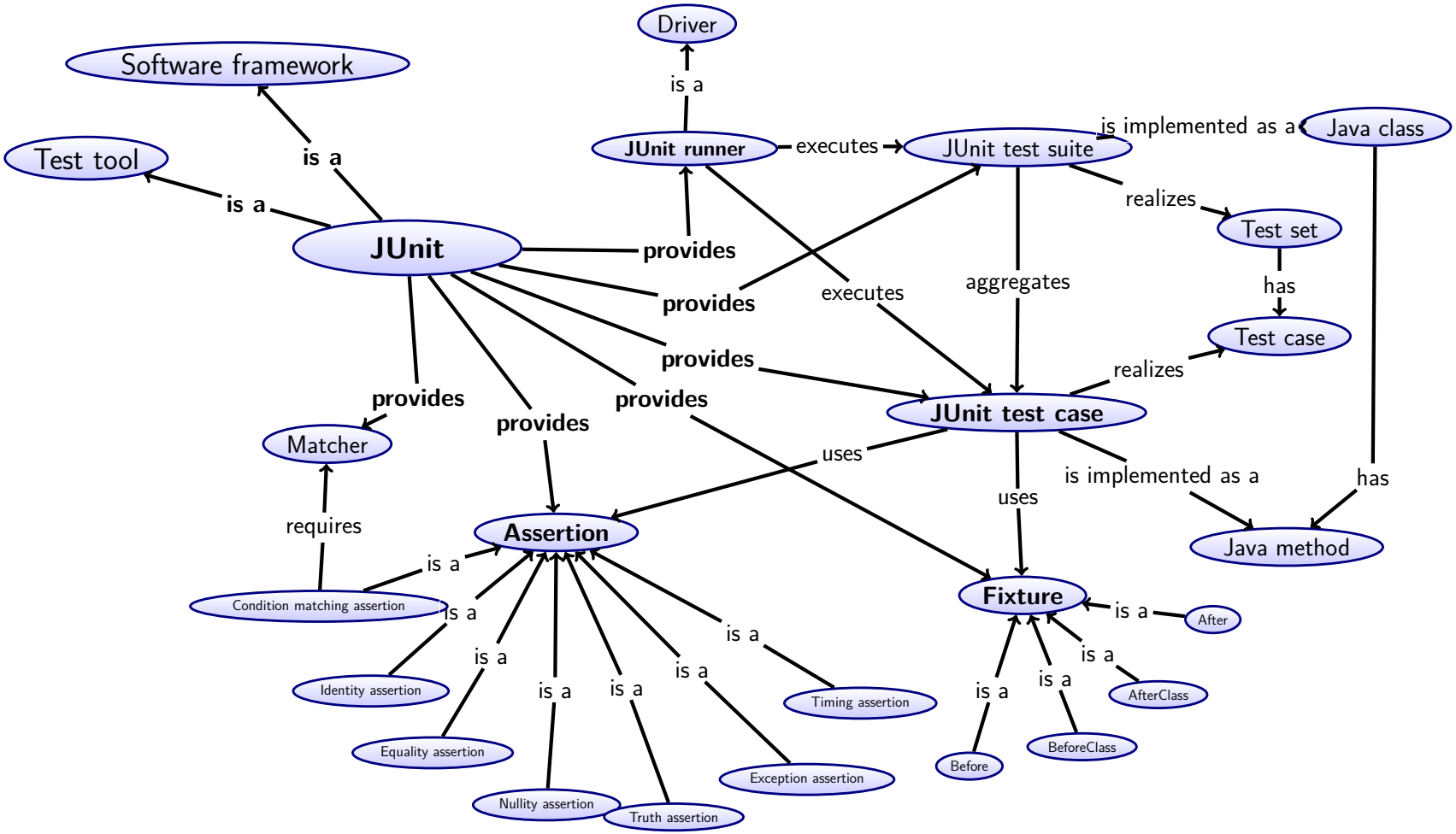
[2]**Institute of Mathematical Sciences and Computing**
University of São Paulo (USP)
São Carlos, SP, Brazil

2017

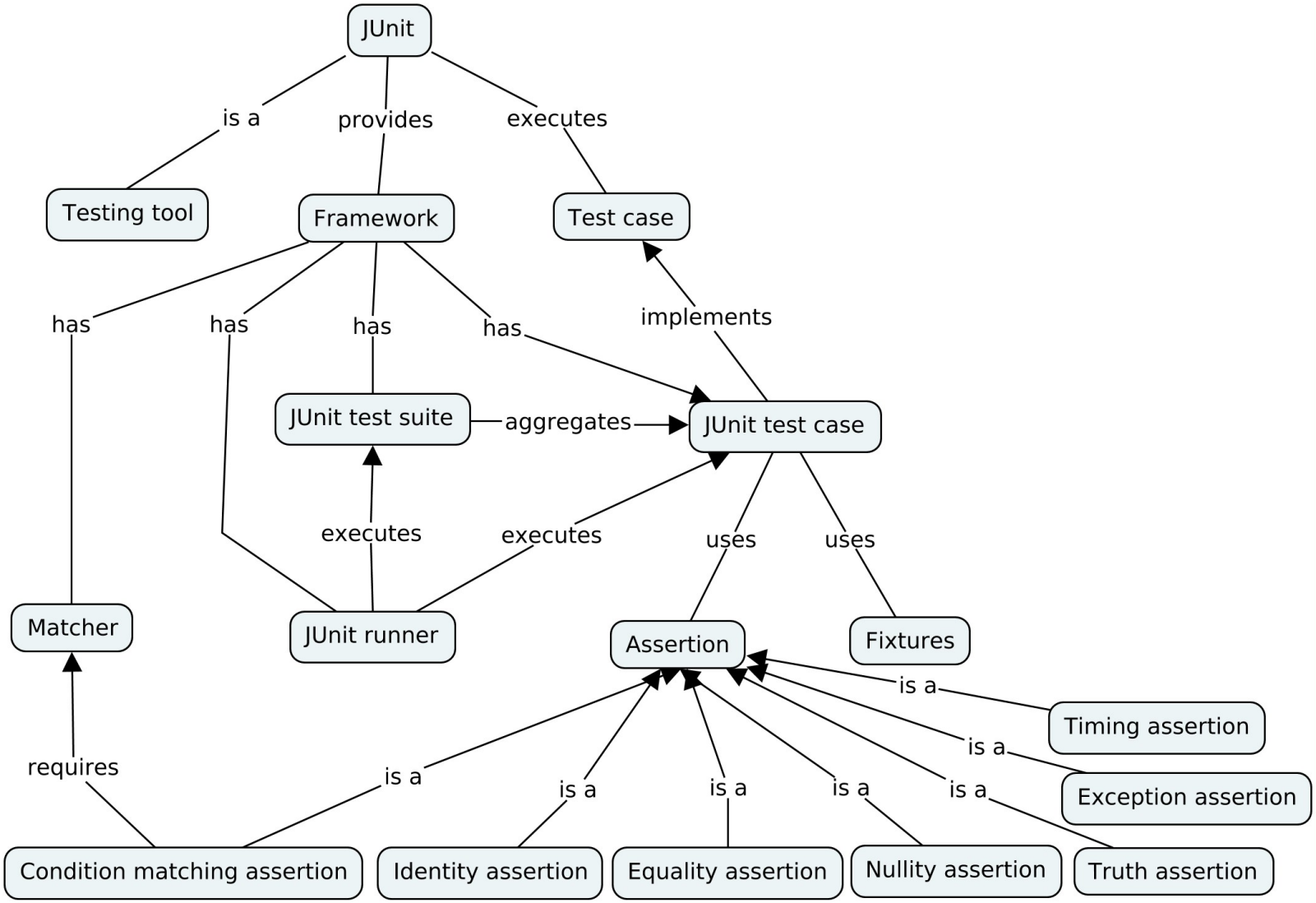
Software Testing



JUnit



JUnit



JUnit

What is it?

JUnit is an open-source framework to provide support for documenting and automating the execution of test sets for Java programs.

General information

- Developed by Kent Beck and Erich Gamma (in 1994).
- Hosted at `https://www.junit.org/` and `https://github.com/junit-team/junit4`.

Features

- Test cases implemented using annotations.
- Useful assertions collection.
- Fixtures enhances the design of test sets.

Example: Identifier

JUnit
Installation (command line)

Requirements

- JUnit requires the Java SDK 1.5 or newer.

1. Download JUnit at `https://github.com/junit-team/junit4/wiki/Download-and-Install`.
   - Current version is 4.12.
   - The application is distributed as two JAR files:
     - `junit.jar`: main JUnit library
     - `hamcrest-core`: library of matchers (optional, only required for `assertThat`)

Classpath configuration

- You can add the library to the CLASSPATH environment variable.

```
Unix:
export CLASSPATH=/opt/junit/junit.jar:
  /opt/junit/hamcrest-core.jar:$CLASSPATH

Windows:
set CLASSPATH=C:\junit\junit.jar;
  C:\junit\hamcrest-core.jar;%CLASSPATH%
```

- You can use the -cp option when running the tests. This is the recommended option!

```
java -cp /opt/junit/junit.jar:/opt/junit/hamcrest-core.jar
<program>
```

Requirements

- Any Eclipse version

For each project you want to use JUnit, proceed as follows:

1. Access the project's properties.
2. Select `Java Build Path` tab on the left.
3. Select `Libraries` tab on the right.
4. Select `Add Library` button on the right of `Libraries` tab.
5. Select `Junit`.
6. Proceed to the next window by pressing the `Next button`.
7. Check if JUnit version is `JUnit 4`.
8. Press `Finish` button.
9. Press `Apply` button.

Is it working?

- To check whether JUnit was correctly installed, you can run the JUnit test suite.
  - The class with all the test cases for JUnit is `org.junit.tests.AllTests`.
  - This class is located at the root of JUnit installation directory.
- Or you may create your own test set! Check the example below.

Example: JUnit shakedown

JUnit test case

Test case

A test case is a pair consisting of test data (a set of values, one for each input variable) to be input to the program and the expected output.

JUnit test case

A JUnit test case is the implementation of a test case as a Java method annotated with `@org.junit.Test`.

How to define a test case

- In general, each test case is defined in a different method within a Java class.
- Test methods neither accept parameters nor return a value.

Example: JUnit test case

How to compile a test case

- To compile a test case, run the Java compiler against the test case file.
  - Remember to include the JUnit library in the classpath.

Example: JUnit test case compilation

JUnit test case
Execution

How to run a test case

- To run JUnit test cases from the command line, run `java org.junit.runner.JUnitCore TestClass1 TestClass2`.

Example: JUnit test case execution

Outcomes

- A test case fails when the generated output value is different than the expected output value.
- A test case succeeds when the generated output value is equal to the expected output value.

How does it detects a failures?

- A JUnit test case fails when an assertion fails (when an `AssertionError` exception is thrown by the test case).

Example: JUnit test case execution outcomes

JUnit test suite and suite runner

Test suite

A JUnit test suite is a class that contains tests from many JUnit test cases classes.

How to define a test suite?

- To create a JUnit test suite, the class (which is usually empty) should be annotated with `@SuiteClasses({TestClass1.class, ...})`.
- To run the JUnit test suite, the class must be annotated with `@RunWith(Suite.class)`

Example: JUnit test suite

Assertion

An assertion is a statement that evaluates as true.

- Assertions work as oracles: they confront obtained and expected outputs, pointing any discrepancies, and enabling the automatic test cases execution.
- JUnit only records failed assertions.

Example: Test case with assertion

JUnit assertions

- Instead of using Java's default assertion mechanism, one can use assertions provided by JUnit.
- JUnit implements several assertions in the class `Assert`:
  - `assertThat`
  - `assertArrayEquals`, `assertEquals`
  - `assertSame`, `assertNotSame`
  - `assertTrue`, `assertFalse`
  - `assertNull`, `assertNotNull`
  - `fail`

Identity assertion

Identity assertions checks if two objects refer to the same object or not.

Methods

- assertSame
- assertNotSame

Example: Identity assertion

Nullity assertion

Nullity assertions check if an object is null.

Methods

- `assertNull`
- `assertNotNull`

Example: Nullity assertion

Equality assertion

Equality assertions checks if the objects are equal (has the same content).

Equality and identity

- Identity assertion implies Equality assertion.

Methods

- `assertArrayEquals`
- `assertEquals`

Example: Equality assertion

Exception assertion

An Exception assertion checks whether an exception is thrown by the test case.

Annotation

- If the JUnit test case expects an exception to be thrown, it must declare the expected exception in the `@Test` annotation, at the `expected` parameter
  - (e.g., `@Test(expected=IndexOutOfBoundsException.class)`.

Example: Exception assertion

Timing assertion

A timing assertion checks if the test case is executed in a given time frame.

Annotation

- JUnit test cases can be annotated with a `timeout` parameter
  - E.g., `@Test(timeout=2000)`
- If the test takes longer than the specified number of milliseconds to run, the test fails.

Example: Timing assertion

Truch assertion

A truth assertion checks if a condition is true or false.

Methods

- assertTrue
- assertFalse

Example: Truth assertion

Condition matching assertion

A condition matching assertion checks whether a given object matches the condition specified by the assertion.

Method

- `assertThat`
  - The AssertThat assertion provides more readable and typeable statements, combinations of any matcher statement, more readable failure messages, and custom matchers.

Example: Condition matching assertion

Fixture

- Fixtures are actions that should be executed before or after a test case (usually to set up pre-conditions).
- It defines a fixed state of a set of objects used as a baseline for running tests.

Why should I use fixtures?

- The purpose of a test fixture is to ensure that there is a well known and fixed environment in which tests are run so that results are repeatable.

Before fixture

Before is a fixture that is used to set up pre-conditions for a test case.

How to use it?

- The Before fixture is created by annotating a method with `@Before`.
- Before fixtures run before a JUnit test case.
- Before fixtures declared in the superclasses will be run before those of the current class.
- No ordering is defined when running Before fixtures declared in the same class.

BeforeClass

BeforeClass is a fixture that is used to set up preconditions for a test set.

How to use it?

- The BeforeClass fixture is created by annotating a method with `@BeforeClass`.
- BeforeClass fixtures run before all the JUnit test cases in a class have been run.
- BeforeClass fixtures declared in the superclasses will be run after those of the current class.
- No other ordering is defined when running BeforeClass fixtures declared in the same class.

After

After is a fixture that is used to cleanup modifications made for or by a test case.

How to use it?

- The After fixture is created by annotating a method with `@After`.
- After fixtures run after a JUnit test case.
- After fixtures declared in the superclasses will be run before those of the current class.
- No ordering is defined when running After fixtures declared in the same class.

AfterClass

AfterClass is a fixture that is used to cleanup modifications made for or by a test set.

How to use it?

- The AfterClass fixture is created by annotating a method with `@AfterClass`.
- AfterClass fixtures run after all the JUnit test cases in a class have been run.
- AfterClass fixtures declared in the superclasses will be run after those of the current class.
- No other ordering is defined when running AfterClass fixtures declared in the same class.

References

AMMANN, P.; OFFUTT, J. *Introduction to software testing*. 1. ed. Cambridge,Reino Unido: Cambridge University, 2008. 344 p. ISBN 978-0521880381. Disponível em: <http://cs.gmu.edu/ offutt/softwaretest/>.

MATHUR, A. P. *Foundations of Software Testing*. 1. ed. [S.l.]: Pearson Education, 2008. 689 p. ISBN 978-8131716601.

Credits

Identifier

- The program determines if a given identifier is valid or not in a variant of Pascal language, called Silly Pascal.
- A valid identifier must begin with a letter and must contain only letter or digits.
- Moreover, it must have at least one character and no more than six characters.

```
package identifier;

import org.junit.Test;
import org.junit.Assert;

public abstract class IdentifierTestSet
{
    protected Identifier id;

    @Before
    public void setUp() {
        id = new Identifier();
    }
}
```

Identifier
Test set 1

```
package identifier;

import org.junit.*

public class IdentifierTestSet1 extends IdentifierTestSet
{
  @Test
  public void validate1() {
    boolean result = id.validateIdentifier("Abcd5");
    Assert.assertEquals(true, result);
  }

  @Test
  public void validate2() {
    boolean result = id.validateIdentifier("x12345");
    Assert.assertEquals(true, result);
  }
}
```

```
package identifier;

import org.junit.*

public class IdentifierTestSet2 extends IdentifierTestSet
{
    @Test
    public void validate3() {
        boolean result = id.validateIdentifier("&123");
        Assert.assertFalse(result);
    }

    @Test
    public void validate4() {
        boolean result = id.validateIdentifier("Inv@lido");
        Assert.assertFalse(result);
    }
}
```

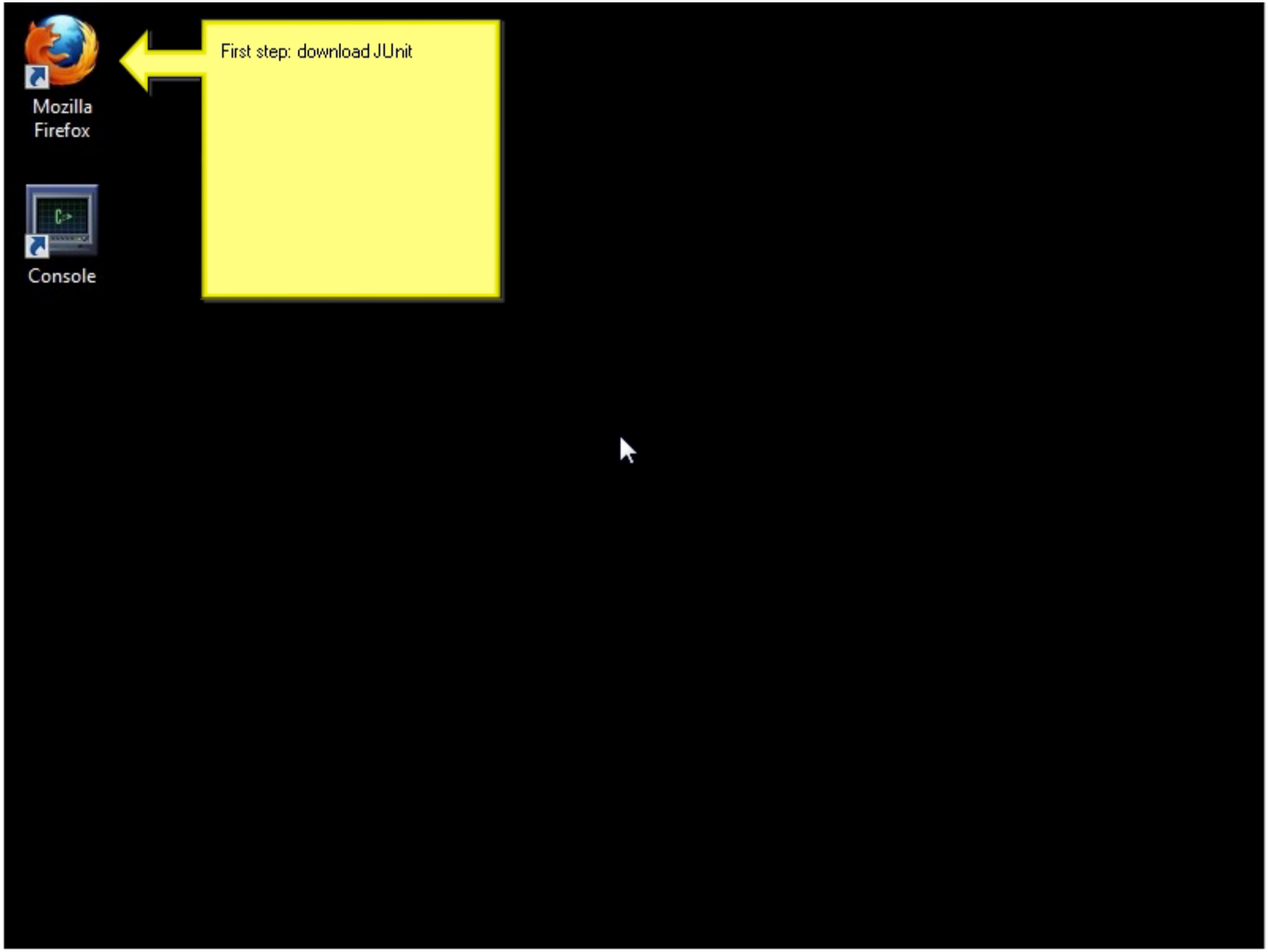
Identifier
Test set 3

```
package identifier;

import org.junit.*;

public class IdentifierTestSet3  extends IdentifierTestSet
{
    @Test
    public void validate5() {
        Assert.assertNotNull(id);
    }

    @Test(expected=IndexOutOfBoundsException.class)
    public void stringException() {
        String str = new String("JUnit Example");
        str.substring(30);
    }
}
```

```
package identifier;

import org.junit.*;

public class IdentifierTestSet4  extends IdentifierTestSet
{
    @Test(timeout=2000)
    public void looping() {
        boolean result = id.validateIdentifier("Abcd5");
        Assert.assertEquals(true, result);
    }

    @Ignore("Out of the program scope")
    @Test(expected=IndexOutOfBoundsException.class)
        public void stringException2() {
        String str = new String("JUnit Example");
        str.substring(30);
    }
}
```

```
package identifier;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
    IdentifierTestSet1.class,
    IdentifierTestSet2.class
    IdentifierTestSet3.class
    IdentifierTestSet4.class
})
public class AllTests
{
}
```

JUnit shakedown



JUnit test case example


```
import org.junit.Test;
import org.junit.Assert;

import java.util.*;

public class ExampleTestCase
{
        @Test
        public void test1() {
                Assert.assertEquals("Test", "Test");
        }

        @Test
        public void test2() {
                List<String> words = new ArrayList<String>()
                words.add("Test");
                Assert.assertNotNull(words.get(0));
                Assert.assertTrue(words.contains("Test"));
        }

        @Test
        public void test3() {
```

Test case compilation


```
# javac \
    -cp /opt/junit-4.8.1/junit-4.8.1.jar
    ExampleTestCase.java
```

Test case execution


```
# java \
    -cp /opt/junit-4.8.1/junit-4.8.1.jar:.
    org.junit.runner.JUnitCore
    ExampleTestCase
```

Test case outcomes


```
$ java \
    -cp /opt/junit-4.8.1/junit-4.8.1.jar:.
    org.junit.runner.JUnitCore
    ExampleTestCase

JUnit version 4.8.1
...E
Time: 0.004
There was 1 failure:
1) test3(ExampleTestCase)
java.lang.AssertionError:
    at org.junit.Assert.fail(Assert.java:91)
    at org.junit.Assert.assertTrue(Assert.java:43)
    at org.junit.Assert.assertTrue(Assert.java:54)
    at ExampleTestCase.test3(ExampleTestCase.java:24)
    [ ... ]
    at org.junit.runner.JUnitCore.run(JUnitCore.java:117)
    at org.junit.runner.JUnitCore.runMain(JUnitCore.java:98)
    at org.junit.runner.JUnitCore.runMainAndExit(JUnitCore.j
    at org.junit.runner.JUnitCore.main(JUnitCore.java:45)

FAILURES!!!
Tests run: 3,  Failures: 1
```

JUnit test suite example
Test suite definition


```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
    ExampleTestCase.class
})
public class AllTests {
}
```

JUnit test suite example
Test suite execution

```
# java \
    -cp /opt/junit-4.8.1/junit-4.8.1.jar:.
    org.junit.runner.JUnitCore
    AllTests
```

Assertion


```
import org.junit.Test;

public class AssertionTestCase
{
  @Test
  public void validate0() {
    assert (2 + 2) == 4;
  }

  @Test
  public void validate1() {
    throw new AssertionError();
  }

}
```

Identity assertion


```
import org.junit.Test;
import org.junit.Assert;

public class IdentityTestCase
{
  @Test
  public void validate0() {
    String s = "test";
    Assert.assertSame(s, s);
  }

  @Test
  public void validate1() {
    String s1 = "test";
    String s2 = "test";
    Assert.assertNotSame(s1, s2);
  }
```

Equality assertion

```
import org.junit.*;

public class EqualityTestCase
{
  @Test
  public void validate0() {
    String s1 = "test";
    String s2 = "test"
    Assert.assertEquals(s1, s2);
  }

  @Test
  public void validate1() {
    String s = "test";
    Assert.assertEquals(s, s);
  }

  @Test
  public void validate2() {
    String[] s1 = {};
    String[] s2 = {};
    Asssert.assertArrayEquals(s1, s2);
  }
```

Nullity assertion


```
import org.junit.Test;
import org.junit.Assert;

public class NullityTestCase
{
  @Test
  public void validate0() {
    String s = null;
    Assert.assertNull(s);
  }

  @Test
  public void validate1() {
    String s = "test";
    Assert.assertNotNull(s);
  }
```

Truth assertion


```
import org.junit.Test;
import org.junit.Assert;

public class TruthTestCase
{
  @Test
  public void validate0() {
    String s1 = "test";
    String s2 = "test"
    Assert.assertFalse(s1 == s2);
  }

  @Test
  public void validate1() {
    String s = "test";
    Assert.assertTrue(s == s);
  }
}
```

Condition matching assertion


```
import org.junit.Test;
import org.junit.Assert;

public class EqualityTestCase
{
    @Test
    public void validate0() {
        String s = "test";
        assertThat(s, eq("test"));
    }

    @Test
    public void validade1() {
        String s = "test";
        assertThat(s, isA(String.class));
    }
}
```

Exception assertion


```
import org.junit.Test;
import org.junit.Assert;

public class ExceptionTestCase
{
  @Test(expected=NullPointerException.class)
  public void validate0() {
    Integer i = null;
    i.toString();
  }
}
```

Timing assertion


```
import org.junit.Test;
import org.junit.Assert;

public class EqualityTestCase
{
  @Test(timeout=1000)
  public void validate0() {
    int counter = 0;
    for (int i = 0; i < 10;) {
       counter += i;
    }
  }
```