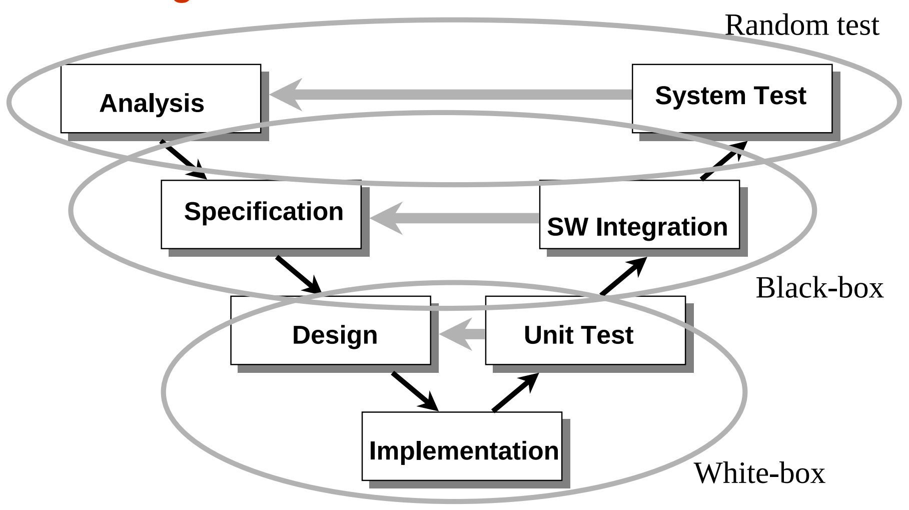
Unit Testing

Testing



"V-lifecycle"

What is a test?

For specification S and program P:

- A **test case** is an input-output pair (i,S(i))
- A **test** is a comparison: S(i)=P(i)
- A **test suite** is a set of test cases which in some sense "cover" the possibilities

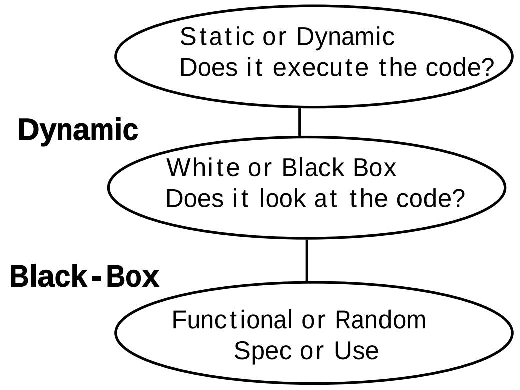
White box

Structural testing which requires internal implementation detail of components/ modules/ objects. In OO terms requires access to **private** fields and methods.

Black box

Functional testing at interface to component/ module / object. In OO terms access only **public** fields and methods.

Kinds of testing



Static Examples:

Type Checking, Inspection, Proof

JUnit - a test framework

Addresses the problem:

"Testing is not closely integrated with development. This prevents you from measuring the progress of development- you can't tell when something starts working or when something stops working. Using JUnit you can cheaply and incrementally build a test suite that will help you measure your progress, spot unintended side effects, and focus your development efforts."

Philosophy:

Develop tests in parallel with application code or before!!

Example

Develop a program to solve the problem of representing arithmetic with multiple currencies.

[12 "UKP"] - twelve UK pounds
[23 "USD"] - twenty three US dollars

Arithmetic between single currencies is trivial, you can just add the two amounts. Simple numbers suffice. So we will start with simple currencies.

Simple Money

```
class Money {
  private int fAmount;
  private String fCurrency;

  public Money(int amount, String currency) {
    fAmount= amount; fCurrency= currency;
  }

  public int amount() { return fAmount; }

  public String currency() { return fCurrency; }

  public Money add(Money m) {
    return new Money(amount()+m.amount(),currency());
  }
```

equals, hashCode, toString

```
    public boolean equals(Object anObject) {
      if (! (anObject instanceof Money) )
         return false;
      Money m= (Money)anObject;
      return m.currency().equals(currency())
        && amount() == m.amount();
    }

    public int hashCode() {
      return currency().hashCode() + amount();
    }

    public String toString() {
      return "["+amount()+" "+currency()+"]";
    }
}
```

test suite using JUnit

```
package money;
import junit.framework.*;

public class MoneyTest extends TestCase {

  public MoneyTest(String name) {
    super(name);
  }
  /**
  *   TestRunner.run uses introspection to find all the test methods.
  *   These must be public & begin with "test"
  */
  public static void main(String args[]) {
    junit.textui.TestRunner.run(MoneyTest.class);
  }
```

the test fixture - set up code

```
private Money f12UKP,f14UKP;

/**
* setUp constructs the context for the tests
* called before each test
*/
protected void setUp() {
   f12UKP= new Money(12, "UKP");
   f14UKP= new Money(14, "UKP");
}

/**
* similarly, clearDown can be used to clean up
* after a test
*/
```

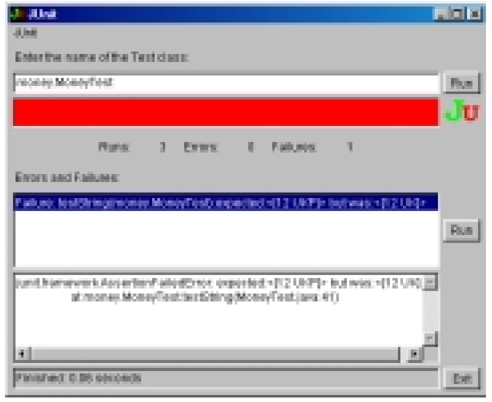
the tests

```
public void testEquals() {
  assert(!f12UKP.equals(null));
  assert(!f12UKP.equals(f14UKP));
  assertEquals(f12UKP,new Money(12,"UKP"));
}

public void testAdd() {
  assertEquals(f12UKP.add(f14UKP),new Money(26,"UKP"));
}

public void testString() {
  assertEquals(f12UKP.toString(),"[12 UKP]");
}
```

running the tests



Enter the name of the Test class:

money.MoneyTest

Runs: 1 Errors: 0 Failures: 1

Errors and Failures:

Failure: testString(money.MoneyTest): expected:<[12 UKP]> but was:<[12 UK]>

```
junit.framework.AssertionFailedError: expected:<[12 UKP]> but was:<[12 UK]
        at money.MoneyTest.testString(MoneyTest.java:41)
```

Finished: 0.05 seconds

adding methods to Money

```
public Money subtract(Money m) {
  return add(m.negate());
}

public Money negate() {
  return new Money(-amount(),currency());
}

public boolean isZero() {
  return amount() == 0;
}
```

adding tests to MoneyTest

```
public void testNegate() {
  assertEquals(f12UKP.negate(),
               new Money(-12,"UKP"));
}

public void testSub() {
  assertEquals(f12UKP.subtract(f14UKP),
               new Money(-2,"UKP"));
  assert((f12UKP.subtract(
               new Money(12,"UKP"))).isZero());
}
```

extending to multiple currencies:

```
package money;
public interface IMoney {

  IMoney add (IMoney m);
  /**
  * helper functions for double dispatch
  */
  IMoney addMoney(Money m);
  IMoney addMoneyBag(MoneyBag m);

  IMoney subtract (IMoney m);

  IMoney negate ();

  boolean isZero();

}
```

We add a new class MoneyBag to handle bags of different types of currencies and a class MoneyBagTest to test the new implementation.

In addition, we have to modify Money to implement IMoney and to be aware of multiple currencies:

```
public IMoney add(IMoney m) {
  return m.addMoney(this);
}
public IMoney addMoney(Money m) {
  if (m.currency().equals(currency()) )
    return new Money(amount()+m.amount(), currency());
  return new MoneyBag(this, m);
}
public IMoney addMoneyBag(MoneyBag s) {
   return s.addMoney(this);
}
```

Combining test suites

```
package money;
import junit.framework.*;
public class TestAll extends TestCase {
   public TestAll(String name) {
      super(name);
   }
   public static TestSuite suite() {
      TestSuite all = new TestSuite();
      all.addTest(new TestSuite(MoneyTest.class));
      all.addTest(new TestSuite(MoneyBagTest.class));
      return all;
   }
   public static void main(String args[]) {
       junit.textui.TestRunner.run(suite());
   }
}
```

Summary

Whenever, we modify code, we add new tests and rerun old tests to make sure that nothing has been “broken”.

We may have to modify old tests that fail because the test is not compatible with the modified code - as opposed to detecting an error in the code.

JUnit is a framework that simplifies writing and running tests.

Note:

Whenever you consider using a `System.out.println`, write a test!

Resources

- **JU**nit at www.junit.org
- Money is a modified version of the example distributed with **JU**nit. It is available on the **doc** lab system.