

Automated Unit Testing Frameworks

Bo Simonsen

bo@geekworld.dk

Department of Computer Science
The University of Copenhagen

March 28, 2008

Introduction

Motivation

Test frameworks

JUnit

JUnit - More features

JUnit - Running the tests

CppUnit

UnitTest++



The straight forward way of automated testing

Automated test can be done "manually" in this way

- Write a test program
- Exit with status 0 if everything went all right
- Use a shell script like this one:

```
#!/bin/sh
```

```
for i in `ls test*.cpp|cut -f1 -d '.'|xargs`  
do  
  ./$i  
  if [ $? != 0 ];  
  then  
    echo "$i failed!"  
  else  
    echo "$i ok.."  
  fi  
done
```



Why using test frameworks?

- The test programs is identical in style
- Easy access to:
 - Exception checking
 - Checking of time constraints (usefull for realtime apps)
 - ...
- Test code which is using frameworks are generally smaller than test code which doesn't use frameworks.

JUnit

- Unit Testing Framework for Java
- Father of CPPUnit, PHPUnit, PyUnit, ...
- Details:
 - For Java 5, implementation is done using the Java annotation (source code meta data)
 - For older versions of Java, the realization class is an extension of the abstract class TestCase
 - It's backward compatible (no need for rewriting old tests)

A simple example

```
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class MyTest
{
    private int a = 0;

    @Before public void setUp() {
        a = 5;
    }

    @Test public void simpleTest() {
        assertEquals(a, 5);
    }
}
```

More features

- Cleanup can be done by

```
import org.junit.After;
@After public void cleanItUp () {
    ..
}
```

- Testing exceptions

```
@Test(expected=ArithmeticException.class)
public void divideByZero() {
    int n = 2 / 0;
}
```

- Realtime constraints

```
@Test(timeout=500)
public void thisIsCritical() {
    /* Do some realtime critical operations */
}
```

- We can ignore a test case by doing this (usefull if you write the test before the actual implementation)

```
import org.junit.After;
@Ignore public void testThatWillFail() {
    assertEquals(a, 0);
}
```

Running the tests

```
$ export JDEVEL="/home/bo/slides"  
$ export CLASSPATH="$CLASSPATH:$JDEVEL/junit-4.4.jar  
:JDEVEL"  
$ javac MyTest.java  
$ java org.junit.runner.JUnitCore MyTest  
JUnit version 4.4  
..  
Time: 0.103  
  
OK (2 tests)
```

And if something goes wrong (along comes a full stack trace)

FAILURES!!!

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

CppUnit

- CppUnit is a port of JUnit
- Works by simple inheritance (and a few macros), e.g.

```
class IntTest : public CppUnit::TestFixture {
private:
    int a;
public:
    IntTest() : a(0);

    void setUp() {
        a = 5;
    }
    void TestIt() {
        CPPUNIT_ASSERT( a, 5 );
    }
};
```

- And from the main program the test class should be used in this way

```
CppUnit::TestCaller<IntTest> test( "IntTest",  
    &IntTest::TestIt );  
CppUnit::TestResult result;  
test.run( &result );
```

- You can classify the tests into "Suites". Eg. you can create a suite which test X methods, and then you run this suite the X methods is tested.
- CppUnit needs alot of code even for doing simple tests. Creating a suite needs more code than a slide

UnitTest++

- UnitTest++ is a macro driven test framework.
- It's much more simple than CppUnit.
- A simple example

```
// test.cpp
#include <UnitTest++.h>

TEST(FailSpectacularly)
{
    CHECK(false);
}

int main()
{
    return UnitTest::RunAllTests();
}
```

A test suite

A test suite can be easily built by simply using the "SUITE" macro around the tests. E.g.

```
SUITE(YourSuiteName)
{
    TEST(YourTestName)
    {
    }

    TEST(YourOtherTestName)
    {
    }
}
```

Checks

- Basic assertion
`CHECK(false);`
- Equality check
`CHECK_EQUAL(10, 20);`
- Floating point check
`CHECK_CLOSE(3.14, 3.1415, 0.01);`
- Array equality check
`CHECK_ARRAY_EQUAL(oned, oned, 2);`
- Floating point array check
`CHECK_ARRAY_CLOSE(oned, oned, 2, 0.00);`
- Two dimensional array check
`CHECK_ARRAY2D(twod, twod, 2, 3, 0.00);`
- Exception check
`CHECK_THROW(throw TestException(), TestException);`

Time constraints

```
TEST(YourTimedTest)
{
    // Lengthy setup...

    {
        UNITTEST_TIME_CONSTRAINT(50);

        // Do time-critical stuff
    }

    // Lengthy teardown...
}
```

Conclusion

Here is a my oppinion about each framework:

- JUnit seems to be the most mature of all the testing frameworks, and the tests can written very simple.
- CppUnit requires alot of code, which can make the code quite hard to read.
- UnitTest++ gives you the opportunity to make very clear and very simple test code.

Questions?

????????????????

References

- <http://www-128.ibm.com/developerworks/java/library/j-junit4.html> - IBM - An early look at JUnit 4
- <http://unittest-cpp.sourceforge.net/UnitTest++.html> - UnitTest++ in brief
- http://cppunit.sourceforge.net/doc/lastest/cppunit_cookbook.html - CppUnit Cookbook