

# Assignment 2 (tools)

Jacob de Fine Skibsted `jacob@defineskibsted.net`

## 1. Unit testing

The purpose of unit testing is to validate the individual components of source code automatically. These components are typically defined as different classes when unit testing is applied to object oriented programming. The method involves writing separate applications which explores the functionality of the components and ensures that the functionality is maintained every time an update is applied to the component. Thus, when a programmer modifies some source code, the test application is run to check if the desired functionality of the component is maintained. Intuitively, this method involves changing not only the source code but also the test application if the functionality of the component is changed and/or extended. Otherwise the test application could erroneously report an error even though the component does perform the desired functionality. As a result, unit testing can be quite time-consuming due to the fact that two code bases must be maintained.

On the other hand, when it works, unit testing is an invaluable tool for determining if some update works as intended. This is especially true when code refactoring is performed. Furthermore, when working in open source communities, where a number of developers from all over the world contribute to some code base, having some way of determining if an update to some smaller component influences the working of others is important. This is where maintaining a test suite becomes handy.

As a result, a number of frameworks for performing unit testing has been proposed.

### 1.1 *cppunit*

cppunit<sup>1</sup> is a unit testing framework used in a C++ programming environment. It is based upon the popular Junit<sup>2</sup> framework developed for java programming environments. A thorough documentation for cppunit can be at <http://cppunit.sourceforge.net/doc/lastest/index.html>.

#### 1.1.1 *Installation and prerequisites*

cppunit is developed for Microsoft windows (95/98/NT/2000/XP) and all POSIX compliant operating systems. It includes an MSDN integration for quick reference to the cppunit functionality.

To install cppunit using linux, the source code can be retrieved from <http://sourceforge.net/projects/cppunit>. Installation is done by executing the commands:

```
shell>./configure
shell>make
shell>make install
```

This will compile and install the framework libraries.

#### 1.1.2 *Usage of cppunit*

As stated, cppunit is a framework consisting of a number of classes which can be instantiated, derived and modified. cppunit, as most other unit testing frameworks uses a test fixture as a basis for running test cases. The purpose of the test fixture is to ensure that there is a well known and fixed environment in which tests are run, so that results are repeatable.

In order to group the different test cases defined by a test fixture, cppunit uses a test suite. Test suites can contain other test suites in order to create a large hierarchy of test cases, which comes in handy when the code base becomes large.

As a tool for executing tests, cppunit uses a test caller to execute a single test case. In extension to this, a test runner is used to execute the complex tests defined by a test suite. The test suite itself, uses test callers to perform the individual test cases.

---

<sup>1</sup> <http://sourceforge.net/projects/cppunit>

<sup>2</sup> [www.junit.org](http://www.junit.org)

### 1.1.3 An example

Tests are created using a fixture which is a class for creating a common environment for a set of test cases. It is constructed by inheriting the class `CppUnit::TestFixture`, and must implement the procedures `setUp()` and `tearDown()`. These procedures may be used to perform initialization and clean-up. Each test case which is to be examined by the fixture is defined as public procedures. An example is given below:

```
#include <iostream>
#include <stack>
#include "TestCase.h"

class StackFixtureTest : public CppUnit::TestFixture {
private:
    std::stack<int> S;
public:
    void setUp() {}
    void tearDown() {}

    void TestStack()
    {
        CPPUNIT_ASSERT(S.empty() == true);
        S.push(100);
        CPPUNIT_ASSERT(((unsigned int) S.size()) == 1);
        CPPUNIT_ASSERT(S.top() == 100);
    }
};
```

For each of the test cases defined in a fixture, a test caller template must be instantiated. The instantiation of the `TestCaller` template class is done using the address of the test case procedure as given below:

```
CppUnit::TestCaller<StackFixtureTest> test( "Testing of a stack",
                                             &StackFixtureTest::TestStack );
```

Thus, when the test caller is run the specified method will be executed. In order to perform a number of individual tests, a test suite is used inside the fixture. Thus, the class `TestSuite` is used in the fixture, again in a separate procedure. Thus the code given below is added to the constructed fixture.

```
static CppUnit::Test *suite()
{
    CppUnit::TestSuite *ts = new CppUnit::TestSuite( "StackTest" );
    ts->addTest( new CppUnit::TestCaller<StackFixtureTest>(
                                                "testStack",
                                                &StackFixtureTest::TestStack ) );

    return ts;
}
```

```
}
```

Thus, a caller to every test case is added to the test suite in order to maintain a structure which separates the different fixtures from each other. Every fixture has its own test suite which defines which of the test cases are to be performed by the suite.

When the test is to be run, a `TestRunner` class must be used which can run a number of different suites. Thus, a test runner wraps all the suites defined in the different fixtures:

```
CppUnit::TextUi::TestRunner runner;
runner.addTest( StackFixtureTest::suite() );
runner.addTest( NextFixture::suite() );
runner.run();
```

## *1.2 Conclusion*

cppunit offers a simple and very extensive way to perform unit testing of C++ source code. Due to this complexity, this essay focuses on a subset of the features. However, the documentation of cppunit found at <http://cppunit.sourceforge.net/doc/lastest/> is good. Unfortunately, the amount of books written on the subject is few, but as cppunit is a port of Junit, most of the material on Junit can be helpful.