Introduction to Automated Unit Testing (xUnit)

Brian Nielsen
Arne Skou
{bnielsen | ask}@cs.aau.dk
Conventional Test Execution

- Ad hoc manner
  - Manual stimulation & observation
  - E.g. adding a function to a module, which runs tests on the module’s functions
  - Uncommenting or deleting test code / drivers / printf / #ifdefs
  - Assert and debug builds
  - Home-brewed test-code and test runners
Automated Testing

- “Code that isn’t tested doesn’t work”
- “Code that isn’t regression tested suffers from code rot (breaks eventually)”
- “If it is not automated it is not done!”

- Boring
- Repetitive
- Necessary
- Error-prone (for humans)
- Better done by you than your users
What is a testing framework?

- A test framework is a software tool for writing and running unit-tests
- provides reusable test functionality which:
  - Enables automatic execution for regression tests
  - Is standardized
  - Easy to use
  - GUI-test case browser/runner
  - Test report generation
What is a testing framework

- Programmer Friendly
  - Test cases written in same language as implementation
  - Well integrated in IDE’s
What is xUnit?

- A set of “Frameworks” for programming and automated execution of test-cases
- X stands for programming language
  - Most Famous is J-UNIT for Java
  - But exists for almost all programming languages
    - C-unit, Cpp-Unit, DUnit, JUnit NUnit, ...
- A framework is a collection of classes, procedures, and macros
Basic Use of FrameWork

- cunit.lib
- myUnitTests.c
- C-compiler
- myUnitTests.exe
- Test-report.xml
Concepts

- **Assertions**
  - Boolean expression that compares expected and actual results
  - The basic and smallest building-block
  - General: **ASSERT** (expected, actual)

- **Test Case**
  - A class that extends “TestCase”s
  - A composition of concrete test procedures
  - May contain several assertions and test for several test objectives
  - E.g. all test of a particular function

- **Test Suite**
  - Collection of related test cases
  - Can be executed automatically in a single command
xUnit

Object

YourClass

setUp
tearDown

TestCase

TestYourClass

SomeMethod1

Test

SomeMethod2

Test

SomeMethod1 and SomeMethod2
class ClassifyTriangle {

    public enum TriangleKind { invalidTriangle, equilateralTriangle, isoscelesTriangle, scaleneTriangle};

    public TriangleKind classifyTriangle(int a, int b, int c) {
        ... return kind;
    }

    public String checkTriangle(String[] args) {
        ...
    }
}
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;
public class ClassifyTriangleTest extends TestCase {
    protected void setUp() {} protected void setUp() {}

    public void testEquilateral() {
        ClassifyTriangle c=new ClassifyTriangle();
        assertEquals(equilateralTriangle, c.classifyTriangle(5,5,5));
        //add more tests here
    }

    public void testCommandLine() {
        ClassifyTriangle c=new ClassifyTriangle();
        assertEquals("Error Code 40!\n", c.checkTriangle({"-1", "Hello World", "-1"});
        //add more tests here
    }

    public static void main (String[] args) {
        junit.textui.TestRunner.run(ClassifyTriangleTest.class);
    }
}
Test Reports

C:\NovoUnitTest\TriangleDemo\cppunitDemo>Debug\cppunitDemo.exe
.F...

c:\novounittest\triangledemo\testtriangle\testtriangle.cpp(30):Assertion
Test name: TriangleTests::validClassification
equality assertion failed
- Expected: 1
- Actual : 4

Failures !!!
Run: 4   Failure total: 1   Failures: 1   Errors: 0

Test Report

FailedTests

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>FailureType</th>
<th>Location</th>
<th>Message</th>
</tr>
</thead>
</table>
| 1 | TriangleTests::validClassification Assertion | Line #30 in c:\novounittest\triangledemo\testtriangle\testtriangle.cpp | equality assertion failed
|   |                           |                   |                                   | - Expected: 1
|   |                           |                   |                                   | - Actual : 4

Statistics

<table>
<thead>
<tr>
<th>Status</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tests</td>
<td>4</td>
</tr>
<tr>
<td>FailuresTotal</td>
<td>1</td>
</tr>
<tr>
<td>Errors</td>
<td>0</td>
</tr>
<tr>
<td>Failures</td>
<td>1</td>
</tr>
</tbody>
</table>
Test Runner XML file

CUnit - A Unit testing framework for C.
http://cunit.sourceforge.net/

Running Suite Suite_1

Running test sample gcd test case ... Passed

<table>
<thead>
<tr>
<th>Type</th>
<th>Total</th>
<th>Run</th>
<th>Succeeded</th>
<th>Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suites</td>
<td>1</td>
<td>1</td>
<td>- NA -</td>
<td>0</td>
</tr>
<tr>
<td>Test Cases</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Assertions</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

File Generated By CUnitv2.1-0 at Thu Mar 15 16:14:33 2007
Advice: xUnit style

- Test cases exhibit isolation
  - Independent of other tests
  - Execution order irrelevant
- Set up an independent environment
  - setUp / tearDown methods scenario
- Each test case performs a distinct logical check
  - ⇒ one or few asserts per test method
  - BUT consider amount of test code declarations to be written (when a assert fails the test method is stopped and no further asserts are checked).
- Test expected errors and exceptions
Advice: xUnit style

- Make them fast;
  - If slow, developers won’t run them.
    - Smoke test suites
    - Complete test suites

- All developers must know about them;
  - Everyone who touches the code must run the tests.
  - Add to common code-repository

- Make test-code as nice and readable as implementation code
  - Documentation, Maintainability
Advice: Daily Builds

- Regression testing “must” be automated
  - This requires they report pass/fail results in a standardized way

- Daily *(Nightly)* builds and testing
  - Clean & check out latest build tree
  - Run tests
  - Put results on a web page & send mail (if tests fail)
Advice: Version Control

- Keep test code in a separate directory
- Keep both tests-sources and implementation-source in version control
- Don’t checkin unless version passes all tests
Advice: Application

- Design and program for testability
- Directly applicable to
  - Pure function libraries
  - API
- (With some footwork also user interfaces, network-, web-, and database applications)
Advice: xUNIT principles

- Write test suite for each unit in the program.
- All test can be executed (automatically) at any time.
- For each program modification all tests must be passed before the modification is regarded as complete - regression testing
- Test First – implement later!
- Originally based on “eXtreme Programming” principles:
  - Lightweight software development methodology
    - by programmers for programmers
- TDD (Test Driven Development) cycle
  1. Write test case, and check it fails
  2. Write the new code
  3. Check that the test passes (and maybe refactor, re-test)
Conclusions

- Code that isn’t tested doesn’t work”
- “Code that isn’t regression tested suffers from code rot (breaks eventually)”
- A unit testing framework enables efficient and effective unit & regression testing
- Use xUNIT to store and maintain all the small tests that you write anyway
- Write tests instead of playing with debugger and printf – tests can be automatically repeated