Introduction to Unit Testing

Presented by Chris Mennie

Administrivia

- Send me group information
  - Tell me if you're still looking for a group
- Project documents have been updated
  - Still in the works, but some things clarified
  - Plus FAQ in interface doc
- Nortel Labs
  - Open? Everything ok?
- Electronic assignment submission

What Are We Doing?

- Testing small portions of our code
  - Unit could be a function, class, group of “things”
- Want to be automated
  - Hand testing very prone to error
  - “printf” testing ugly and not useful in the long term
- Want repeatable and deterministic tests
  - Should be easily run from central point
  - Provide continue confidence while coding

General Process

- Write tests first
  - Not always possible
  - Document what the tests will be
- Encodes requirements
  - Gives guidance when there isn't proper design
- Hard to do as a first step, but well worth

Test Organisation

- Start with basic test
  - A function to call
- Extend that to a Fixture
  - Framework for tests with setup / teardown
- Add tests to Fixture
- Build Suite of Fixtures (or other Suites)
- Run test Suite

Using JUnit

- Simple test
  - ```
    public void testSimpleAdd() {
    ```
  - ```
        Money m12CHF= new Money(12, "CHF");
    ```
  - ```
        Money m14CHF= new Money(14, "CHF");
    ```
  - ```
        Money expected= new Money(26, "CHF");
    ```
  - ```
        Money result= m12CHF.add(m14CHF);
    ```
  - ```
        assertTrue(expected.equals(result));
    ```
  - ```
    }
    ```
    ```

Using JUnit

- Fixture
  - ```
    public class MoneyTest extends TestCase {
        private Money f12CHF;
        private Money f14CHF;
        private Money f28USD;

        protected void setUp() {
            f12CHF= new Money(12, "CHF");
            f14CHF= new Money(14, "CHF");
            f28USD= new Money(28, "USD");
        }
    }
    ```
    ```

Using JUnit

- Suite and TestRunner

```
import junit.framework.*;
public class Driver {
    public static void main(String args[])  {
        junit.textui.TestRunner.run(suite());
    }
    public static Test suite() {
        TestSuite suite = new TestSuite(MoneyTest.class);
        return suite;
    }//main
}//Driver
```

Using JUnit

- Don't have to put all tests into Suite
- Can set up Suite manually to choose test subset
- See JUnit webpage for more information

Using CppUnit

- Simple Test
  - ```
    class ComplexNumberTest : public CppUnit::TestCase {
    ```
  - ```
    public:
    ```
  - ```
      ComplexNumberTest( std::string name ) : CppUnit::TestCase( name )
    {}
    ```
  -
  - ```
      void runTest() {
    ```
  - ```
        CPPUNIT_ASSERT( Complex (10, 1) == Complex (10, 1) );
    ```
  - ```
        CPPUNIT_ASSERT( !(Complex (1, 1) == Complex (2, 2)) );
    ```
  - ```
      }
    ```
  - ```
    };
    ```

Using CppUnit

- Fixture
  - `class ComplexNumberTest : public CppUnit::TestFixture {`
  - `private:`
  - `  Complex *m_10_1, *m_1_1;`
  - `protected:`
  - `  void setUp()  {`
  - `    m_10_1 = new Complex( 10, 1 );`
  - `    m_1_1 = new Complex( 1, 1 );  }`
  - `  void tearDown()   {`
  - `    delete m_10_1;`
  - `    delete m_1_1;  }`
  - `};`

Using CppUnit

- Suite and TestRunner
  - `public: <in ComplexNumberTest>`
  - `static CppUnit::Test *suite() {`
  - `CppUnit::TestSuite *suiteOfTests = new CppUnit::TestSuite ( "ComplexNumberTest" );`
  -
  - `suiteOfTests->addTest( new CppUnit::TestCaller<ComplexNumberTest> ("testEquality", &ComplexNumberTest::testEquality ) );`
  - `suiteOfTests->addTest( new CppUnit::TestCaller<ComplexNumberTest> ("testAddition", &ComplexNumberTest::testAddition ) );`
  - `return suiteOfTests;`
  - `}`

Using CppUnit

- Suite and TestRunner

```
int main( int argc, char **argv)
{
  CppUnit::TextUi::TestRunner runner;
  runner.addTest( ComplexNumberTest::suite() );
  runner.run();
  return 0;
}
```

Test Architecture

- What goes into a fixture?
- How many suites?
- Tend towards grouping similar things together
  - May not happen for this deliverable
- Test organization should be reasonable
  - Won't be too picky this deliverable, but do your best

Testing Call Processing (SE-3)

- So far we've looked at functional tests
- Haven't considered multi-process / IPC
- A little harder, but generally the same sort of thing
- This is where the <Test...> messages are useful
  - ie: They have nothing to do with fault detection

Testing Call Processing (SE-3)

- What do we want?
  - Single point, running same sort of tests
- What do we have to work with?
  - The <Test...> messages
- What do we need?
  - To know the state of the phone process
  - A hook into the phone process's connection to the interface server

Testing Call Processing

- My suggestion
  - Add another thread to your phone process
    - Has a pointer to the phone object / connection
    - Connects to a “test server”
  - Create a new “test server” process
    - Accepts a connection from phones
    - Contains the tests to be run
    - Asks phones to send <Test...> messages
    - Asks phones for their states / relevant information

Testing Call Processing

- “Test Server” runs tests as we've been discussing
- Tests look like
  - sendDigitPress(phone1, “9”)
  - sendDigitRelease(phone1, “9”)
  - ...
  - state = getState(phone1)
  - assertTrue(state == expectedState)

Testing Call Processing

- New threads in phone process provide
  - thin abstraction to phone connection
  - access to call processing state
- Ideally has zero to minimal affect on main call processing thread

Testing Call Processing

- Of course there's other ways
- Testing stuff could be bundled in with regular signaling
  - “Test server” “calls” phones (pull vs. push)
- Could write an interface simulator
  - Needs a few extra signals
- The thing to take away is phone processes can be tested automatically

Final Comments

- Complete, working, examples will be posted
- Read the webpages, play with the frameworks