

# A Brief Introduction to Unit Testing and PyUnit

Chris Simpkins

# Overview

- Unit Testing Basics: What, Why, When
- How: Debugging an IM Client Bug with PyUnit
- Unit Test Writing Tips and Resources

# Unit Testing Basics

- Smoke tests: most common kind of test.
- Functional tests: done by QA to test functionality according to a test plan based on requirements and design specs.
- Unit tests: done by developers to test specific code. Typically “white box” testing.
- Essential part of Extreme Programming and other agile methods.

# Why Write Unit Tests

- Increase developers' confidence in code. If someone challenges your work, you can say “the tests passed.”
- Avoid regression. If unit test suite is run frequently, you know when new code breaks old code.
- If you write tests first, you know when you're done, i.e., when the tests pass.
- Encourages minimal interfaces and modularity.

# When to Write/Run Unit Tests

- Always!
- Before you check code into repository, so you know your code works.
- Before and after refactoring, so you know your redesign doesn't break anything.
- Before debugging, to ease the process and help you know when you're done.

# A PyUnit Test Suite

```
import unittest

class MainFrameTest(unittest.TestCase):

    def setUp(self):
        """called before each test."""

    def tearDown(self):
        """called after each test."""

    def testUpdateChats(self):

    def suite():
        return unittest.makeSuite(MainFrameTest, 'test')

if __name__ == "__main__":
    unittest.main()
```

# My IM Client Bug

- Bug scenario:
  - A,B, C are online.
  - A chats with B
  - C gets a chat window showing a chat with A and B
- Problem: internal conversations data structure adds all chats, including irrelevant ones.

# First Step in Debugging: Create Unit Test to Expose Bug

```
def testUpdateChats(self):
    conversations = {1:["Me", "Beavis"]}
    self.mainFrame.updateChats(conversations)
    self.assertEqual(self.mainFrame.conversations, conversations)

    conversations[2] = ["Me", "Butthead"]
    self.mainFrame.updateChats(conversations)
    self.assertEqual(self.mainFrame.conversations, conversations)

    # Save a copy of the conversations before adding a
    # conversation in which Me is not a part.
    expectedConversations = copy.copy(conversations)
    conversations[3] = ["Beavis", "Butthead"]
    self.mainFrame.updateChats(conversations)
    # Make sure the new conversation is not recorded internally as
    # one of Me's chats.
    self.assertEqual(expectedConversations,
                     self.mainFrame.conversations)
```

# Test Should Fail if Bug Present

```
=====
FAIL: testUpdateChats (__main__.MainFrameTest)
-----
```

```
Traceback (most recent call last):
```

```
File "/Users/Shared/jythonRelease_2_2alpha1/Lib/unittest.py", line 213, in __call__
    testMethod()
```

```
File "gui-test.py", line 39, in testUpdateChats
```

```
    self.assertEqual(self.mainFrame.conversations, expectedConversations)
```

```
File "/Users/Shared/jythonRelease_2_2alpha1/Lib/unittest.py", line 286, in failUnlessEqual
    raise self.failureException, \
```

```
AssertionError: {3: ['Beavis', 'Butthead'], 2: ['Me', 'Butthead'], 1: ['Me', 'Beavis']} != {2: ['Me', 'Butthead'], 1: ['Me', 'Beavis']}
```

```
-----
Ran 1 test in 1.157s
```

```
FAILED (failures=1)
```

# Fix Bug, Test Passes

```
[csimpkins@lawn-199-77-214-101: src]$ jython guitest.py
```

```
.
```

```
-----  
Ran 1 test in 1.543s
```

```
OK
```

Now we also have a regression test to catch the bug if it reappears.

# Test Writing Tips

- Make code modular: use interfaces/template classes/abstract base classes.
- Use mock objects to inspect behavior of object you're testing and to stand in for "heavy" objects, e.g., you don't want to do network I/O in a unit test.
- Modular, loosely coupled interfaces make mock objects possible.
- Excessive coupling is enemy of unit testing.

# Resources

- <http://pyunit.sourceforge.net/>
- <http://pyunit.sourceforge.net/pyunit.html>
- <http://www.junit.org/>