SoCal Code Camp 2006, Fullerton, January 21, 2006

Agile Testing with Python Test Frameworks

Grig Gheorghiu
Avamar

What is agile testing?

- Practice TDD
- Short feedback loop between developers and testers
  - find bugs that matter early rather than late
  - collaborate with programmers to improve testability
  - shift from last line of defense to service provider
- Focus on providing value
  - fluid roles: do whatever job needs to be done
  - toolsmith role: automate what you can, deliver in less than 40 hours
  - spend time and resources only on things that provide information
- Involve customers early in writing business-facing tests
- Deploy automated regression testing
- Conduct sessions of exploratory testing

Resources:
James Bach's site (exploratory testing and much more): http://www.satisfice.com
Brian Marick's Agile Testing site: http://testing.com/agile
agile-testing Yahoo group: http://groups.yahoo.com/group/agile-testing
Elizabeth Hendrickson article on Agile Testing: http://www.qualitytree.com/ruminate/011905.htm
TDD portal: http://www.testdriven.com

Python as an agile language

- SIMPLICITY
  - Clean and simple syntax reads like pseudo-code
  - Built-in high level data types
  - Naturally object-oriented, encourages code reuse
- COMMUNICATION
  - Powerful yet simple idioms
  - Lets you code and doesn't get in your way
  - Standard coding style enforced by significant whitespace
  - "Executable documentation" via doctest

Python as an agile language (cont.)

- FEEDBACK
  - Dynamic, interpreted: short development cycle, fast feedback loop
  - Interactive shell session provides instantaneousfeedback
  - Unit test frameworks (unittest, doctest, py.test) enable frequent feedback
- COURAGE
  - Can write code that is simple, provides quick feedback and can be easily understood by your peers
  - Simple act of coding in Python produces pure pleasure...so:
  - Can have the guts to throw away code that doesn't work and start afresh

Ron Jeffries:
"""
Extreme Programmingis a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringingthe whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation.
"""

Unit testing with unittest

- Agile Testing practice: "Practice TDD"
- Port of jUnit to Python (un-"pythonic"?)
  - easy to use by people familiar with the xUnit frameworks
  - test classes need to inherit from unittest.TestCase
  - API can get in the way and can make the test code intent hard to understand
- Provides fixture management via setUp and tearDown
- Strong support for test organizationand reuse via test suites
- Custom assertions: assertEqual, assertTrue
- Flexibility in test execution via command-line arguments

Resources:
pyUnit home page: http://pyunit.sourceforge.net/pyunit.html

```
import unittest

class TestSort(unittest.TestCase):
    def setUp(self):
        self.alist = [5, 2, 3, 1, 4]

    def test_ascending_sort(self):
        self.alist.sort()
        self.assertEqual(self.alist, [1, 2, 3, 4, 5])

def suite():
    suite = unittest.TestSuite()
    suite.addTest(unittest.makeSuite(TestSort))
    return suite

if __name__ == "__main__":
    unittest.main()
    #unittest.TextTestRunner(verbosity=2).run(suite())
```

Unit testing with doctest

- "Executable documentation" or "Literate testing"
  - no API to remember: copy and paste output from shell session
  - perfect way to keep documentation in sync with code
- Can be combined with epydoc to produce "Agile Documentation"
- No provisions for test fixture management or test suite organization (except when used in conjunction with unittest)
- Output matching mechanism mandates that output must be the same on every run
- Common usage
  - testing small functions with few loops
  - avoiding the overhead of unittest in setting up/tearing down state

Resources:
doctest documentation: http://docs.python.org/lib/module-doctest.html
epydoc home page: http://epydoc.sourceforge.net/

```
def test_ascending_sort():
    """
    >>> a = [5, 2, 3, 1, 4]
    >>> a.sort()
    >>> a
    [1, 2, 3, 4, 5]
    """
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Unit testing with py.test

- No API
  - test functions/methods start w. test_ and test classes with Test
  - save test code in files that start with test
  - use normal "assert" statements
- setup_[method|class|module] hooks provide test fixture management at method, class and module-level
- Collection mechanism provides test suite organization
- Customized traceback/assertion output; strong debugging support
- A lot of magic behind the scenes; can obscure the tool's intent
- Available via subversion; active development; expect many changes

Resources:
py.test home page: http://codespeak.net/py/current/doc/test.html
py lib home page: http://codespeak.net/py/current/doc/why_py.html

```
class TestSort:
    def setup_method(self, method):
        self.alist = [5, 2, 3, 1, 4]

    def test_ascending_sort(self):
        self.alist.sort()
        assert self.alist == [1, 2, 3, 4, 5]

# py.test -v test_sort.py
inserting into sys.path: /usr/local/dist-py
============================= test process starts ==============================
testing-mode: inprocess
```

```
executable  : /usr/local/bin/python  (2.4.0-final-0)
using py lib: /usr/local/dist-py/py
initial testconfig 0: /usr/local/dist-py/py/test/defaultconfig.py/.
================================================================================
0.000 ok test_sort.py:5       TestSort.test_ascending_sort()
0.000 ok test_sort.py:9       TestSort.test_custom_sort()
0.000 ok test_sort.py:23      TestSort.test_sort_reverse()
0.007 ok test_sort.py:28      TestSort.test_sort_exception()
```

Agile development tools

- PyLint: Python source code analyzer
  - looks for bugs and signs of poor quality
  - checks if modules satisfy coding standards
- coverage: statement coverage for Python code
  - accumulates coverage data over many runs
  - generates coverage reports
- buildbot: continuous integration
  - automates compile/test cycle
  - can be used with any projects, not just Python
  - status delivery through web pages, email, IRC
- trac: wiki, project management, bug tracking
  - bugs/tasks/enhancements entered as tickets
  - tickets associated with milestones
  - integrates with subversion

Resources:
PyLint: http://www.logilab.org/projects/pylint
coverage: http://www.nedbatchelder.com/code/modules/coverage.html
buildbot: http://buildbot.sourceforge.net
trac: http://edgewall.com/trac/

Homegrown test automation framework

- Main functionality my team needs to test
  - backup a set of files/directories/mail messages/databases
  - restore them
  - verify that the restored data is identical to the original data
- We encapsulated this functionality in a BRV class (Backup-Restore-Validate)
  - main methods: backup, restore, restore_validate, generate_data
  - overridden as necessary by subclasses that need more specific behavior
  - BRV class provides a main_loop method that runs backup/restore/validate
  - most of the time only generate_data needs to be overridden
  - BRV methods extended with special functionality (Exchange, Oracle RMAN)

```
def main_loop(self):
    self.test = Test.Test('EXECUTABLE.' + self.__class__._name_)
    try:
        # start timer
        self.timer.start("total")

        # generate data
        if self.test.config.isset("generate"):
            self.generate_data()
            if self.test.config.isset("norun"):
                self.timer.stop("total")
```

```
                    return

            # backup
            self.backup()

            # restore
            self.restore()

            # validate
            self.restore_validate()

            # end timer
            self.timer.stop("total")

            self.test.add_memo("test took times of: %s" % self.timer.timestr())
        except:
            # If we have an uncaught exception here we should log this test as INVALID
(exceptiontype, exceptionvalue, exceptiontrace) = sys.exc_info()
            msg = "Uncaught exception %s in basic::main_loop" % str(exceptiontype)
            self.test.test({"message": msg})
            self.test.set_passfail('INVALID')
            self.test.log_result()
```

Homegrown test automation framework

- Test result management encapsulated in a Test class
  - logs results to stdout, log files, database
  - HTTP POST mechanism: Apache Web server with Firebird DB backend
- Generation of large data sets for load testing
  - 100s of Active Directory users/Exchange mailboxes (win32com, win32net)
  - 1000s of Exchange folders/messages (CDO/MAPI, ADO)
  - large Oracle databases (cx_Oracle)
  - large SQL Server databases (mxODBC)
  - large DB2 databases (pyDB2)
  - large Firebird databases (KInterbasDB)

STAF/STAX

- Our goal: automated regression test on many clients/platforms
- From IBM: automated test distribution, execution and reporting
  - STAF agent runs on all the machines STAF testbed
  - STAX Service machine coordinates agents, gathers all results and logs
  - Commands are sent to STAX Service host via XML "job" files
  - Jython code can be embedded in script elements inside XML
  - commands are executed on agents via process elements
  - function definitions and calls
  - iterate, parallelliterate, timer constructs
  - logging and notification constructs
- Global Python variables via SCRIPTFILE and SCRIPT commands
- Java GUI tool: STAX Job Monitor
- Used by Linux Test Project (http://ltp.sourceforge.net)

Sample command sent to STAX Service machine:

```
STAF mgmt1 STAX
EXECUTE FILE /QA/STAF/stax_jobs/client_test_harness.xml
MACHINE mgmt1
```

```
SCRIPTFILE /QA/STAF/stax_jobs/global_vars.py
JOBNAME "CLIENT_TEST_HARNESS"
SCRIPT "VERSION='1.0.2'"
CLEARLOGS Enabled
```

Sample STAX job file:

```
<script>
  HFSADDR = 'dpe03'
  HARNESS_TIMER_DURATION = '60m'

  clients_os = {'neon':'unix',
                                    'sunv2401':'unix',
                                    'ibmp6151':'unix',
                                    'copper':'win',
                                    'dellwin2ks2':'win'
                                    }

  pylts_path = {'unix': '/data01/qa/pylts',
                                'win' : 'C:/qa/pylts'
                                }

  tests_unix = [
  [ 'unix_perms', 'avtar/snapup_unix_perms.py' ],
  [ 'commonality', 'avtar/snapup_commonality.py' ],
  [ 'weird_names', 'avtar/snapup_weird_names.py' ],
  ]

  tests_win = [
  [ 'srv_unicode_names', 'avtar/srv_unicode_names.py' ],
  ]
</script>

<defaultcall function="Main"/>

<function name="Main">
  <sequence>
    <import machine="'neon'" file="'/vortex/pub/QA/STAF/stax_jobs/log_result.xml'"/>
    <call function="'ClientTestHarness'">
      [clients_os, pylts_path, tests_unix, tests_win]
    </call>
  </sequence>
</function>

<function name="ClientTestHarness">
  <function-list-args>
    <function-required-arg name='clients_os'/>
    <function-required-arg name='pylts_path'/>
    <function-required-arg name='tests_unix'/>
    <function-required-arg name='tests_win'/>
    <function-other-args name='args'/>
  </function-list-args>
  <paralleliterate var="machine" in="clients_os.keys()">
    <sequence>
    <script>
    os_type = clients_os[machine]
    tests = {}
    if os_type == 'unix':
         tests = tests_unix
    if os_type == 'win':
         tests = tests_win
    </script>
    <iterate var="test" in="tests">
      <sequence>
```

```
        <script>
        test_name = machine + "_" + test[0]
                  </script>
        <testcase name="test_name">
          <sequence>
            <script>
            cmdline = pylts_path[os_type] + "/" + test[1] + " --hfsaddr=%s" % HFSADDR
            </script>
                          <timer duration = "HARNESS_TIMER_DURATION">
                                  <process>
                                    <location>machine</location>
                                    <command>'python'</command>
                                    <parms>cmdline</parms>
                                    <stderr mode="'stdout'" />
                                    <returnstdout />
                                  </process>
                          </timer>
            <call function="'LogResult'">machine</call>
          </sequence>
        </testcase>
      </sequence>
    </iterate>
    </sequence>
  </paralleliterate>
</function>
</stax>
```

Automated regression test scenario

- Nightly build completion message sent to distributionlist
- .procmailrcfile triggers Python script which sends STAX job file to STAX Service machine
- Script queries central log file for the new job
- Script stops querying on line containing 'Stop|JobID: jobID'
- Script sends message with job log and test count (overall, pass and fail)

Acceptance testing with FitNesse/PyFIT

- Agile testing practices
  - involve customers in writing business-facing tests
  - automated acceptance testing
- FitNesse is based on Ward Cunningham's FIT
  - tests as HTML tables with inputs and expected outputs
  - FitNesse specific: Web server/Wiki, Test Suites, SetUp/TearDown pages
- FIT/FitNesse test the business logic below the GUI
  - enforce good code design practices (MVC)
  - create test interfaces for better code testability
- Test tables drive the AUT via fixtures written in Java, C#, Python
  - fixture is thin wrapper called by the FitNesse framework
  - fixture delegates all work to the AUT

Resources:
FitNesse home page: http://www.fitnesse.org
FitNesse fixtures: http://fitnesse.org/FitNesse.WritingAcceptanceTests
PyFIT available in the files section of http://groups.yahoo.com/group/fitnesse

FitNesse/PyFIT: ColumnFixture tables

- ColumnFixture: similar to SQL insert/update/select single row
- Posting a new entry to a blog:

BloggerFixtures.PostNewEntry			
title	content	valid?	num_entries?
Entry #1 Title	Entry #1 Content	true	1
Entry #2 Title	Entry #2 Content	true	2

- Deleting an entry by its index:

BloggerFixtures.DeleteEntry	1
valid?	num_entries?
true	2

Resources:
FitNesse ColumnFixture: http://fitnesse.org/FitNesse.ColumnFixture

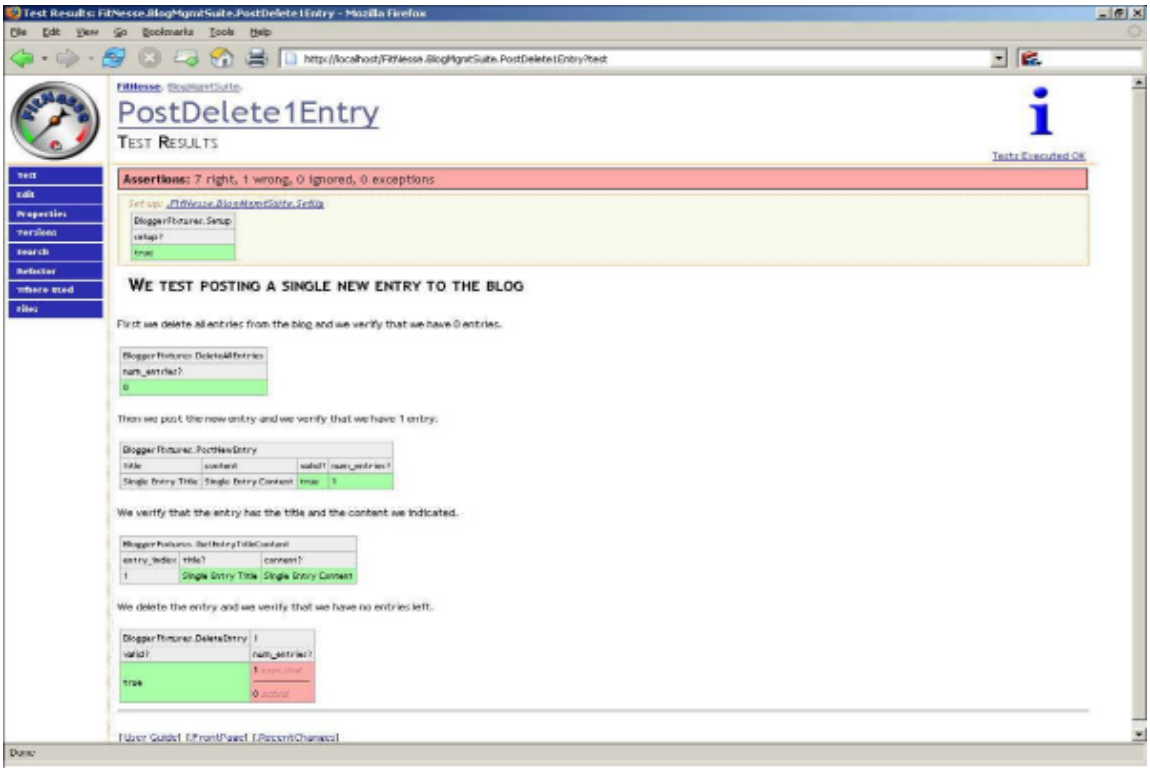
FitNesse/PyFIT: RowFixture tables

- RowFixture: similar to SQL select *
- List index, title and content for all entries in a blog:

BloggerFixtures.ListAllEntries		
entry_index	title	content
1	Entry #2 Title	Entry #2 Content
2	Entry #1 Title	Entry #1 Content

Resources:
FitNesse RowFixture: http://fitnesse.org/FitNesse.RowFixture

FitNesse/PyFIT: Sample test run



FitNesse/PyFIT: Other types of fixtures

- ActionFixture: commands that emulate user interface
- HtmlFixture: assertions about the structure of HTML pages
- CommandLineFixture: execute shell commands in multiple threads
- SummaryFixture: generates report of all tests on a page

Web application testing

- Two categories of test tools
- Tools that simulate browsers ("Web protocol drivers")
  - implement HTTP request/response protocol
  - some parse HTML into DOM
  - Twill, mechanize, webunit (Python)
  - HttpUnit, jWebUnit, HtmlUnit (Java)
  - WWW:mechanize (Perl), WebUnit (Ruby)
- Tools that automate browsers ("Web browser drivers")
  - drive real browsers via JavaScript or COM calls
  - examples: Selenium, Pamie (IE), Jssh (Mozilla), Watir (Ruby/IE)

Resources:
Twill: http://darcs.idyll.org/~t/projects/twill/README.html
mechanize: http://wwwsearch.sourceforge.net/mechanize
webunit: http://mechanicalcat.net/tech/webunit
Selenium: http://selenium.thoughtworks.com/index.html
Pamie: http://pamie.sourceforge.net
Jssh: http://www.croczilla.com/jssh
Watir: http://rubyforge.org/projects/wtr

Browser simulator: twill

- Can be used as a domain specific language via a command shell (twill-sh)
- Very easy to deploy and use (command-line based)
- Perfect for automating form handling
- Implements commands such as:
  - go -- visit the given URL
  - find -- assert that the page contains a regular expression
  - showforms -- show all of the forms on the page
  - formvalue --- set the given field in the given form to the given value
  - submit -- click a submit button

Resources:
Titus Brown's twill home page: http://www.idyll.org/~t/www-tools/twill/
Wikipedia RSS feed based on twill: http://www.deheus.net/petrik/blog/post/68/

Browser automation: Selenium

- Uses real browser to play back testing scripts
- Based on JavaScript: cross-platform and cross-browser
- Tests can be written as HTML tables, similar to FitNesse
- Browser can also be driven via scripts (Python, Ruby, Perl)
- "TestRunner" mode: static HTML, JS, CSS pages on Web site under test
  - Ajax functionality can be tested via mouse and keyboard events
- "Driven" mode: standalone server (Twisted-based)
  - bypasses JavaScript Cross-Site Scripting security limitations
  - allows testing of any Web site
  - browser driven by scripts
- Selenium Recorder available as Firefox extension

Resources:
Selenium home page: http://openqa.org/selenium/
Selenium Recorder: http://seleniumrecorder.mozdev.org/

Python GUI test tools

- pyGUIUnit: unit testing for GUIs created with PyQt
- dogtail: created by RedHat engineers on linux
  - Uses the X11 accessibility framework (AT-SPI)
- Marathon: Java Swing GUI testing

Resources:
pyGUIUnit: http://pyguiunit.sourceforge.net
dogtail: http://people.redhat.com/zcerza/dogtail/
Marathon: http://marathonman.sourceforge.net

Jython in the testing world

- Number one contender in the Dynamic Java arena (scripting the JVM)
- Combines agility of Python with easy access to Java libraries
- Guido van Rossum (Python creator): "Testing is a popular area for Jython"
- Java test tools can be easily driven from within Jython
  - Web app testing with HttpUnit from Jython
- Many test tools use Jython as the test scripting language
  - The Grinder v3 (distributed peformance/load/stress test framework
  - TestMaker (distributed Web app testing framework)
  - Marathon (automated Java GUI testing)
  - MaxQ

Resources:
Tim Bray post on Dynamic Java: http://www.tbray.org/ongoing/When/200x/2004/12/08/DynamicJava
The Grinder v3: http://grinder.sourceforge.net/g3/whats-new.html
TestMaker: http://pushtotest.com/Downloads/downloadtmdoc.html
HttpUnit: http://httpunit.sourceforge.net/index.html

Conclusions

- Bret Pettichord: "Homebrew test automation"
- Agile toolsmith: deliver something useful in less than 40 hours
  - No "automation cathedral"
  - Automation driven by the day-to-day test process
  - Automation serving specific immediate needs rather than future needs
- Huge variety of open source test tools available
- Use high-quality scripting language
  - provides prompt feedback
  - enhances productivity
  - makes it easy to do the simplest thing that works

Resources:
Bret Pettichord's "Homebrew test automation" class notes:
http://www.io.com/%7Ewazmo/papers/homebrew_test_automation_200409.pdf
Danny Faught and James Bach: "Not your father's test automation"
(go to http://www.stickyminds.com in the Articles section)

Conclusions (cont.)

- "Holistic" testing:
  - Unit testing with unittest, doctest, py.test
  - Nightly regression testing with STAF/STAX
  - Business logic acceptance testing with PyFIT/FitNesse
  - Web app. GUI acceptance testing with twill and Selenium
- Python/Agile Testing blog: agiletesting.blogspot.com
- Python Testing Tools Taxonomy: http://pycheesecake.org/wiki/PythonTestingToolsTaxonomy
- SoCal Piggies Python Interest Group: http://www.socal-piggies.org