New Tools for Testing Web Applications with Python

presented to PyCon2006
2006/02/25

Tres Seaver
Palladion Software
tseaver@palladion.com
Test Types / Coverage

- Unit tests exercise components in isolation
- Integration tests exercise “assemblies”
- Functional tests exercise “slices” of the system, according to usage
- System tests exercise the configured system as a whole
- The further we go “up”, the poorer our coverage (generally)
Qui custodiet custodiens?

- Tests verify system functionality
  - who verifies tests?
- Nearer the “surface” of an application, user verification becomes more important
- Traditional spellings are aimed at programmers, not users (“whitebox”)
- “Blackbox” testing aimed at users (or QA)
New Testing Technologies

- **Doctests**
  - “whitebox” developer documentation
  - Unit, functional, or integration testing

- **Funkload**
  - “blackbox” testing at the system level
  - Functional testing, load, and stress testing over HTTP

- **Selenium**
  - “blackbox” browser testing
  - Simulates user interaction inside a “real” browser
  - Tests as specifications, from POV of the user
Doctest example

- Tests specify behavior from programmer's POV
- Storytelling, with executable examples

Using mylib
==========

First, we create an instance of the Foo class:
>>> from mylib import Foo
>>> foo = Foo('This is a Foo')

Now, we can query the title:
>>> foo.title
'This is a Foo'
Funkload Features

- Mixes 'unittest' and 'webunit' to emulate a browser from within Python
- Records test scenarios using 'TCPWatch'
- Able to generate “random” input data
- Can be driven from doctest
- Reuse “functional” tests for benchmarking
- Generates ReST or HTML reports with benchmarking data
Funkload Example

- **Iterate over a set of URLs**

```python
import unittest
from funkload.FunkLoadTestCase import FunkLoadTestCase

class SimpleTest(FunkLoadTestCase):
    def test_simple(self):
        SERVER_URL = 'http://localhost:8080/site'
        PAGES = ('index.html', 'about.html', 'login.html')
        for i in range(10):
            for page in PAGES:
                page_url = '%s/%s' % (SERVER_URL, page)
                self.get(page_url, description='Get %s' % page)

if __name__ == '__main__':
    unittest.main()
```
Funkload Testrunners

- **Standard testrunner**
  - `fl-run-test test_simple.py`
  - Supports both pyUnit and doctest formats

- **Benchmarking testrunner**
  - `fl-run-bench test_simple.py SimpleTest.test_simple`
  - Saves profile information in XML file for later processing
Funkload Benchmarking Output

- Benchmark analysis
- fl-build-report funload.xml
- Generated ReST / HTML
- Can be converted to PDF
Browser Testing Addresses Gaps

- Web applications are increasingly pushing more behavior into the browser
  - “AJAX” (Javascript + XML/RPC)
  - “Deferred page assembly”
- Traditional testing cannot exercise this functionality well
- Server-side testing which “emulates” browsers may yield false confidence
Other Advantages

- Cross-browser compatibility tests
  - Browsers are a major source of bugs!
- Test specifications users understand
  - Shared understanding increases acceptance, productivity
  - “FIT” project results
- Bug reporting
  - Blue sky: record user reproducing bug, generate test case
Anatomy of a Selenium Test Case

- Each test case is a simple HTML page, containing a 3-column table
  - First row is ignored
  - Rows consist of triples: VERB | TARGET | DATA
  - Each row is either an “action” or an “assertion”
  - Triples use a FIT-inspired language, “Selenese”
Running a Test Case

- Testrunner sets up “application-under-test” in bottom frame
- As tests run, testcase rows turn green / red
- Tests can be single-stepped, including interaction with browser
Authoring Selenium Test Cases

- Authoring tests can be specification
  - “Fleshing out” use cases
  - Can be done in advance, e.g. using mockups

- Simple HTML format, easy to manage in text editor
  - or with tools like Composer
Recording Selenium Test Cases

- Mozilla / Firefox extensions allow recording / editing test cases
  - [http://www.augure.com/dev/SeleniumEditor.xpi](http://www.augure.com/dev/SeleniumEditor.xpi)
- zope.testrecorder works in IE as well:
  - [http://svn.zope.org/zope.testrecorder/](http://svn.zope.org/zope.testrecorder/)
- Solutions which use 'TCPWatch' unsatisfactory, as they can't seen user "gestures", only HTTP wire traffic
Resources

- Zelenium (Zope integration), http://www.zope.org/Members/tseaver/Zelenium
- Funkload site, http://funkload.nuxeo.com/
Q & A

- Tres Seaver, tseaver@palladion.com