Agile Documentation
(*using tests as documentation*)

Grig Gheorghiu
Avamar

PyCon 2006, Feb. 26, Addison, TX

Documentation is no fun

- “No man but a blockhead ever wrote except for money”
  -- Samuel Johnson

Documentation is no fun

- This makes me a blockhead with a blog

Documentation is necessary

- Documentation is not sufficient, but it sure is necessary
- Feedback-driven documentation: twill
  - feedback is one of the main agile tenets
  - documentation needs to adapt to changes “imposed” by users solving real-life problems
  - start “virtuous circle” by writing good documentation
  - note how more and more people use your software because of good documentation
  - write more documentation

Everybody likes stories

- Storytelling can make documentation more exciting for both writers and readers
- Stories provide context and people tend to remember them
- More all-around fun when **stories are tests**
- Stories in agile methodologies
  - Joshua Kerievsky's **storytests**: capturing stories together with acceptance criteria that validate their implementation
  - Mike Cohn's **user stories**: requirements/features expressed as stories with associated tests

Unit tests vs. acceptance tests

- Unit tests: make sure you write the code right
- Acceptance tests: make sure you write the right code

(Robert Martin, aka “Uncle Bob”)

doctest: unit/functional tests as stories

- doctest: “literate testing” or “executable documentation”
- no API; just copy and paste from the Python interpreter prompt
- tests are part of docstrings
- encourages storytelling
- documentation in various formats (HTML, epydoc) can be generated with minimal glue code

FitNesse: acceptance tests as stories

- FitNesse is a functional/acceptance testing framework based on Ward Cunningham's FIT
- tests expressed in high-level business domain language
  - “business-facing” tests vs. “code-facing tests” (unit tests)
  - tests expressed as stories peppered with tables which specify inputs and expected outputs
  - wiki format encourages collaboration between customers, testers and developers in refining business rules
- FitNesse web site contains its own acceptance test suite: http://fitnesse.org/FitNesse.SuiteAcceptanceTests

Django approach: doctest + HTML

- Django model functionality tested with doctest
  - http://code.djangoproject.com/browser/django/trunk/tests/testapp/models/
- doctest docstrings turned into HTML format via a custom test runner
  - http://code.djangoproject.com/browser/django/trunk/tests/runtests.py
- doctests displayed as API usage examples
  - http://www.djangoproject.com/documentation/models/

Ian Bicking's approach: Excel spreadsheets

- acceptance tests specified as Excel tables, with free-form comments that document the requirements
- custom test runner knows how to tie tests into application code and interpret the results

MailOnnaStick approach: FitNesse and doctest+epydoc

- MailOnnaStick: mail search and annotation engine
- “business logic” (back-end) acceptance tests expressed as stories in FitNesse wiki pages
- unit tests written with both nose and doctest
- lesson learned: unit test duplication is sometimes good
- documentation generated with epydoc

MailOnnaStick: FitNesse stories

- acceptance tests written as FitNesse stories
- tests expressed in specific domain language: mailboxes, messages, search results
- tests exercise back-end functionality, bypassing the GUI
- lesson learned: business-logic tests are very robust in the presence of UI changes

MailOnnaStick: doctest + epydoc = unit testing stories

- doctest unit tests written as stories in docstrings
- minimal markup used to separate testing of specific pieces of functionality
  - http://agile.idyll.org/browser/mail-onna-stick/trunk/tests/doctests/test_db.py
- epydoc processing
  - docstrings from test files shown as stories
  - http://agilistas.org/mos/epydoc-html/

MailOnnaStick: test lists

- Question: what unit tests do we have for module M?
- Answer: look at the test list for M
  - test list = set of unit tests for a given module
- test lists automatically generated from doctest docstrings with minimal code (show gen_tlist.py)
- test lists very easy to see via epydoc

MailOnnaStick: test maps

- Question: where exactly in our unit tests does function F get exercised, if at all?
- Answer: look at the test map
  - test map = set of unit test functions that exercise a given application function (“static” coverage analysis)
  - automatically generated using a hacked version of Michael Hudson's docextractor module
  - does not work for methods yet (maybe when the new AST module will be available)
- see at a glance which functionality is not exercised at all in unit tests; useful combined with coverage reports

Q & A

- Contact info:
  - grig@gheorghiu.net
  - http://agiletesting.blogspot.com