Testing Best^H^H^H^H Good Practices

Michael Peters

Plus Three, LP

Types of Testing

- Unit Testing
  - Individual Units
  - Single Perl module or script
- Integration Testing
  - Do all the pieces fit together
  - Groups of related modules and scripts

Types of Testing

- Acceptance Testing (Functional Testing)
  - Do the pieces work the way the client wants them to
  - Sometimes uses a DSL
  - FIT
- Automated Testing
  - No humans involved
    - Time based
    - Change based

TAP

- Test Anything Protocol
  - And we do try to make it anything
- Started in Perl
- Now language agnostic
- IETF Standard WG

TAP

- Why?
  - Standard way to exchange data about what happened
  - Human readable and scannable
  - Machine readable
  - Separate the running from the reporting of the test

TAP

➔What?

```
1..7
ok 1 - use Arcos::DB::County;
ok 2 - create() missing params
ok 3 - create() missing name param
ok 4 - create() missing fips_code param
ok 5 – create() missing state param
ok 6 # SKIP geo-coding dbs missing
ok 7 # SKIP geo-coding dbs missing
```

Testing Styles

- Object Oriented
  - Xunit
  - Test::Class
- Data Driven
  - XML, JSON, YAML, Perl structures
  - Declarative and easy to maintain
- TDD
  - Write tests before code
  - Frankly a real pain, except for bug fixes

Web Testing

- Same basic principles
  - Given some input
  - Verify that output is correct
  - Input is HTTP
  - Output is HTML, XML, JSON, etc
- Test with your web server
  - Otherwise no guarantee that behaviour is the same
  - Slightly harder in initial setup, but worth it in the long run

Web Testing

- Test::WWW::Mechanize + LWP
- Test::HTML::Content
- Selenium
- Test::HTTP
- Test::Image
- Test::Email
- Test::JSON

Web Testing

- Test::WWW::Mechanize + LWP

```
use Test::WWW::Mechanize;
my $mech = Test::WWW::Mechanize->new();
$mech->get_ok('http://127.0.0.1:8080');
my $form = $mech->form_name('some_form');
ok($form, 'this form exists');
my $input = $form->find_input(name => 'some_input');
ok($input, 'input exists in form');
is($input->value, 'expected value');
```

Web Testing

- Test::HTML::Content (w/Mech)

```
use Test::WWW::Mechanize;
use Test::HTML::Content;
my $mech = Test::WWW::Mechanize->new();
$mech->get_ok('http://127.0.0.1:8080');
my $html = $mech->content;
xpath_ok($html, '//form[@name=”foo”]');
xpath_ok($html, '//form[@name=”foo”]/input[@name=”bar”]');
xpath_ok($html, '//form[@name=”foo”]' .
  '/input[@name=”bar” and @value=”expected value”]');
```

Web Testing

- Selenium

```
use Test::WWW:Selenium;
my $s = Test::WWW::Selenium->new(
  host => 'localhost',
  browser => 'firefox',
);
$s->open('http://localhost:8080/');
ok($s->get_text('//form[@name=”foo”]'));
ok($s->get_text('//form[@name="foo"]/input[@name="bar"]'));
ok($s->get_text('//form[@name="foo"]'
  .'/input[@name="bar" and @value="expected value"]'));
```

Web Testing

- Test::HTTP

```
use Test::HTTP;
my $test = Test::HTTP->new();
$test->get(
  'http://127.0.0.1:8080',
   Accept => 'text/html'
);
$test->status_code('200', 'all ok');
$test->body_like(qr/<html/, 'looks like HTML to me');
$test->put(foo => 1, bar => 2);
$test->post(foo => 2, bar => 3);
```

Web Testing

- Test::Image (w/ Mech)

```
use Test::WWW::Mechanize;
my $mech = Test::WWW::Mechanize->new();
$mech->get_ok('http://127.0.0.1:8080/graph.png');
open(OUT, '>foo.gif');
print OUT $mech->content;
my $img = Test::Image->new(
  Image::Imlib2->new('foo.gif')
);
$img->size(400, 200);
$img->pixel(10, 10, 'white');
```

Web Testing

- Test::Email

```
use Test::Email;
my $email = Test::Email->new(\@lines_of_text);
$email->ok({ from => 'admin@foo.com' });
$email->ok({ subject => qr/Howdy \D+/});
my @parts = $email->parts;
$parts[0]->mime_type_ok('text/html');
$parts[1]->mime_type_ok('text/plain');
```

Web Testing

➔Test::JSON (w/ Mech)

```
use Test::WWW::Mechanize;
my $mech = Test::WWW::Mechanize->new();
$mech->get_ok('http://localhost:8080');
is_json(
  $mech->response->header('x-json'),
  {
    success => 1,
    err_msg => 'You dummy',
  }
);
```

Other Useful Perl Testing Modules

- Test::Most
- Test::Differences
- Test::Output
- Test::Fork
- Test::Valgrind

Writing Your Own Testing Modules

- Test::Builder

```
package Project::MyTests;
use Test::Builder;
use Test::More;

sub is_foo {
  my ($self, $text);
  my $test = Test::Builder->new();
  $test->ok($text eq 'foo', "$text is not foo");
}

sub isnt_foo {
  my ($self, $text);
  my $test = Test::Builder->new();
  $test->ok($text ne 'foo', "$text is foo");
}
```

Writing Your Own Testing Modules

- Test::Builder

```
use Project::MyTests;
my $tests = Project::MyTests->new();
my $foo = do_something();
$tests->is_foo($foo);

my $bar = do_something_else();
$tests->isnt_foo($bar);
```

Test Coverage

- How much of my code is covered?
  - modules
  - subs and methods
  - branches and conditionals
- Devel::Cover
  - perl -MDevel::Cover
  - use Devel::Cover
  - ./Build testcover
  - Pretty, interactive HTML reports

Automated Testing

- Do your tests run all the time?
  - Humans are unreliable
  - Failures should be big, obvious andquick
- Time based
  - via cron
  - svn checkout && make smoke_test
- Change based
  - svn hooks
  - github web hooks

Automated Testing

- How do you know something failed?
  - Web page with latest test run
  - Simple email with test output
  - More bells and whistles
    - Smolder
    - Buildbot
    - Cruise Control

Smolder

- Smoke Test Aggregator
- Tries to answer these questions:
  - When did my tests last run?
  - How many passed/failed/skipped?
  - Which tests passed/failed/skipped?
  - What was the last svn # that passed?
  - How has our test suite grown over time?
  - How long does our full test suite take?
  - What color are my eyes?

Questions?