Automated Testing of Large Projects With Perl

Andy Lester
andy@petdance.com
http://petdance.com/perl/

Where we're going

- Strategy
- What & how to test
- 5 things you can do on Monday
- 5 things for next week
- 5 for next month

About TITLEWAVE

- 2/3rds of sales
- Constantly under development
- **90K** lines of Perl & PHP
- **9215** tests (at press time)

Testing Strategy

- Make testing simple
- Test constantly
- Test extensively
- Make testing part of your culture

Make testing simple

- Humans still have to write the tests
- Tests that are a pain to write won't get written
- Tests that aren't understood won't be maintained
- Use a smoke script to allow testing of selected tests

smoke

```
#!/usr/bin/perl

use File::Find::Rule;
use Test::Harness qw(&runtests);

my $rule = File::Find::Rule->new;
$rule->or(
    $rule->new->directory->name('CVS')->prune->discard,
    $rule->new->file->name( '*.t' )
);

my @start = @ARGV ? @ARGV : '.';
for ( @start ) {
    push( @files, (-d) ? $rule->in($_) : $_ );
}

runtests(@files);
```

Running smoke

```
$ smoke HTML.t
HTML....ok
All tests successful.
Files=1, Tests=52, 0 wallclock secs


$ smoke
HTML.........................ok
Images.......................ok
Page.........................ok
Page/Admin...................ok
Page/Curricula...............ok
All tests successful.
Files=5, Tests=210, 10 wallclock secs
```

Test constantly

- Have a smokebot
- Smoke your file as you're building it
- Smoke your individual files before committing

Test extensively

- Test depth
- Test breadth
- Test anything that's ever gone wrong
- Remember: Human testing doesn't scale

Make testing part of your culture

- Don't write code that can't be tested
- Any test that fails must be fixed immediately
- Code reviews must include the corresponding test files
- Everyone on the team adds at least one new test every day
- If you hire, explicitly look for testing experience in your candidates

Ignore efficiency

- Premature (and unnecessary) optimization is the root of all evil.
- Tests run once an hour, when you're not watching them.

Redundancy is good

- Each test adds to your army of tests
- Ignore DRY (Don't Repeat Yourself)

There are no stupid tests

```
$foo->set_wango( 'tango' );

is( $foo->wango(), 'tango' );
```

- Of course it works! It's just an accessor!
- But what if it doesn't? What will you have to go through to find it?

What to test?

- Modules/libraries
- Coding standards & project information
- Application data
- Application logic
- Anything that has ever gone wrong

Modules/libraries

- Simplest way to start
- Commonly done throughout Perl
- Pick a widely-used module with a lot of tests and steal ideas.

Coding standards

- Is all HTML clean?
- Do you use strict & warnings?
- Does each .pm file have a .t?
- Is POD correctly formatted?
- Write tests to verify!

Checking your project

- Define the start of the directory as an environment: we use $TWROOT.
- Find the files to check
  - Watch out for CVS directories
  - File::Find
  - File::Find::Rule is easier in many cases
- Do your checks on each file in the project

Dev/Rules.t

```
# Find all Perl files, but don't look in CVS

my $rule = File::Find::Rule->new;
$rule->or(
  $rule->new->directory->
        name('CVS')->prune->discard,
  $rule->new->file->name( '*.pl','*.pm','*.t' ));

my @files = $rule->in( $base );

for my $file ( @files ) {
    check( $file );
}
```

Dev/Rules.t

```
sub check {
  my $filename = shift;

  my $dispname =
    File::Spec->abs2rel( $filename, $base );

  local $/ = undef;

  open( my $fh, $filename ) or
    return fail( "Couldn't open $dispname: $!" );
  my $text = <$fh>;
  close $fh;

  like( $text, qr/use strict;/,
    "$dispname uses strict" );
  like( $text, qr/use warnings;|perl -w/,
    "$dispname uses warnings" );
} # check()
```

Dev/HTML.t

```
for my $filename ( @files ) {
  open( my $fh, $filename ) or
    fail( "Couldn't open $filename" ), next;
  local $/ = undef;
  my $text = <$fh>;
  close $fh;

  if ( is_php($text) ) {
    ++$php;
    pass( "$dispname (skip)" );
  } else {
    ++$html;
    my $lint = HTML::Lint->new;
    $lint->only_types( HTML::Lint::Error::STRUCTURE );
    html_ok( $lint, $text, $dispname );
  }
}
diag( "$html HTML files, $php PHP files" );
```

Dev/pod.t

```
use Test::More;
use Test::Pod 0.95;

my @files = .... # build file list

plan( tests => scalar @files );

for my $filename ( @files ) {
    pod_file_ok( $filename );
}
```

Application data

- App data is as important as the code itself.
- Catch things DB constraints can't
  - Customer number formats
  - URL validity
  - App-specific data format

Redundant is good

- Constraints might be too expensive
- Constraints might get deleted accidentally
- Test for valid constraints!
  - Try to add a bad foreign key
  - Make sure it fails

Oracle/syspw.t

```
use Test::More 'tests' => 1;
use FLR::DB qw( :sqldo );

FLR::DB::setparms(
  USERNAME => 'SYS',
  PASSWORD => 'CHANGE_ON_INSTALL'
);

eval { sqldo_column('select 1 from dual'); };
my $failed = defined $@;
ok( $failed, 'SYS user should not have default pw' );
```

Oracle/userscheck.t

```
use Test::DatabaseRow;

my @good_users = qw( FOLLETT CTXSYS MAINFRAME );
plan( tests => @good_users + 1 );

use_ok( 'FLR::DB' );
$Test::DatabaseRow::dbh = FLR::DB::dbh();

for my $user ( @good_users ) {
    row_ok(
        table => "DBA_USERS",
        where => [ username => $user ],
        label => "$user exists",
    );
}
```

Web pages & apps

- WWW::Mechanize wraps LWP::UserAgent and HTML::Form
- Lets you think about interactions, not web mechanics
- HTML::Lint checks validity of HTML

Simple page loading

```
use Test::More tests => 10;
use Test::HTML::Lint;
use WWW::Mechanize;

my $a = WWW::Mechanize->new();
isa_ok( $a, "WWW::Mechanize" ) or die;

$a->get( "http://petdance.com/" );
is( $a->status, 200, 'Fetched OK' );
like( $a->title,
  qr/^petdance.com: Andy Lester/,
  "Correct page title"
);
html_ok( $a->content, "Home page HTML" );
```

Following links

```
# Continuing from previous slide...

$a->follow_link( text_regex => qr/resume/ );
ok( $a->success, 'Got resume' );
like( $a->title, qr/Andy Lester.+resume/,
  "Title correct" );
html_ok( $a->content, "Resume HTML" );

$a->follow_link( text_regex => qr/Google Hacks/ );
ok( $a->success, 'Followed Google Hacks' );
like( $a->title, qr/Google Hacks/, "Title correct" );
like( $a->uri, qr[^http://www.oreilly.com],
  "It's on oreilly.com"
```

The results

```
1..10
ok 1 - The object isa WWW::Mechanize
ok 2 - Fetched OK
ok 3 - Correct page title
ok 4 - Home page HTML
ok 5 - Got resume
ok 6 - Title correct
not ok 7 - Resume HTML
#     Failed test (follow.pl at line 23)
# Errors: Resume HTML
#  (26:5) <ul> at (21:7) is never closed
#  (69:63) </a> with no opening <a>
#  (85:5) <td> at (84:2) is never closed
ok 8 - Followed Google Hacks
ok 9 - Title correct
ok 10 - It's on oreilly.com
# Looks like you failed 1 tests of 10.
```

Testing forms

```
use Test::More tests=>10;
use WWW::Mechanize;

my $a = WWW::Mechanize->new();
isa_ok( $a, 'WWW::Mechanize' ) or die;
$a->get( "http://www.google.com/" );
ok( $a->success, "Got Google" );
$a->form_number(1);
$a->field( q => "Andy Lester" );
$a->click( "btnG" );

ok( $a->success, "Got the results page back" );

$a->follow_link( text_regex => qr/petdance\.com/ );
ok( $a->success, "Followed the link" );
is( $a->uri, "http://www.petdance.com/" );
```

Anything that has ever gone wrong

- Many problems aren't your fault, but you still have to deal with them.
- CGI.pm changed the behavior of the input() function.
- PHP broke the sprintf() function.

Dev/perl/CGI.t

```
#!/usr/bin/perl -w

use strict;
use Test::More tests=>4;

use_ok( 'CGI', ':standard' );
use_ok( 'Test::HTML::Lint' );

# CGI 2.91 broke the handling of the <input> tag.
# Make sure we don't run afoul of it.
# Apparently it's been fixed in 2.92.
INPUT: {
  my $text = input( {foo=>'bar'} );
  is( $text, q{<input foo="bar" />},
    "Built expected string" );
  html_ok( $text, "Valid HTML" );
}
```

Dev/php/sprintf.phpt

```
// printf et al broke between PHP 4.2.3 and PHP 4.3.0
// I reported it as bug #22227
//      http://bugs.php.net/bug.php?id=22227
// Closed as being the same as #20108
//      http://bugs.php.net/bug.php?id=20108

require( "Test.php" );

plan( 3 );

diag( "PHP Version " . phpversion() );

$masks = Array( "%-3.3s", "%.3s", "%-.3s" );
$str = "abcdefg";
foreach ( $masks as $mask ) {
    $result = sprintf( "[$mask]", $str );
    is( $result, "[abc]", "[$mask]" );
}
test_end();
```

How do I start?

5 things to do Monday

1. Read Schwern's Test::Tutorial
2. Start a test suite for one module
3. Start a smokebot
4. Start looking at t/*.t for modules you use
5. Spread the gospel

5 for next week

1. Write tests for an entire module
2. Add at least one new test daily
3. Start keeping metrics
4. Memorize Test::More, Scalar::Util and Test::Util. Explore other Test::* modules.
5. Read *The Pragmatic Programmer* and at least one other book on XP or testing.

5 for next month

1. Post your first month of metrics.
2. Write data and application tests.
3. Modify your coding standards or process to explicitly require tests.
4. Look at JUnit to see what you can learn.
5. Create your own domain-specific Test::* module. Post it to CPAN if appropriate.

More information

http://petdance.com/perl/

Test::Tutorial

Schwern's slides:
http://mangonel.guild.net/~schwern/talks/

The Pragmatic Programmer, especially chapter 8

Any good XP intro book, like *Extreme Programming Installed*