Test::Cookbook

Belden Lyman,
shutterstock

YAPC::NA 2012

twitter: @belden

this talk on github:
github.com/belden/test-cookbook

Test Driven Development

I think we generally think about it like this

I think about Test Driven Development like this

i.e. we're taking your code for a test drive

Your tests should give you info

- Right, you get to measure correctness; we get it
- Your test harness can have meta-tests
  - this test has failed if it's slower than 2 standard deviations from the norm
- Let you know when the bad refactoring has happened

Above all, Test Driven Development needs failures to be meaningful

Consider:

```
use Test::More;

my @got = (1, [2..5, {hello     => 'world'}, [8..10]]);
my @exp = (0, [1..4, {goodnight => 'moon' }, [7..10]]);

is_deeply( \@got, \@exp);
__END__
#     Structures begin differing at:
#          $got->[0] = '1'
#     $expected->[0] = '0'
```

More info is better

A simple toolset

- `deep_ok()`
- `set_ok()`
- `xml_ok()`
- `system_ok()`
- `memoized_ok()`
- `stdout_of(&), stderr_of(&)`

slides/simple-toolset.t
slides/xml.t

Tests should be easy to maintain

- Tests in a loop are easy to write...
  - maintaining testcounts gets annoying
  - understanding failure modes becomes hard
  - chances are each iteration isn't really a unique test anyway

mock objects

It's not that hard to roll your own mock object library.

- We're Perl developers; we don't always get to be purely TDD
- Help you pin down the behavior of a legacy system

Monkeypatching

```
#!/usr/bin/perl

{
  package some;
  sub where { 'beyond the sea' }
}

is( some->where, 'beyond the sea' );

{
  no warnings 'redefine';
  local *some::where = sub { 'over the rainbow' };
  is( some->where, 'over the rainbow' );
}
```

resub - monkeypatching *delivered*

dependency injection

This allows users of your code to specify what objects should be created, and how.

work on slides/mock_object.t

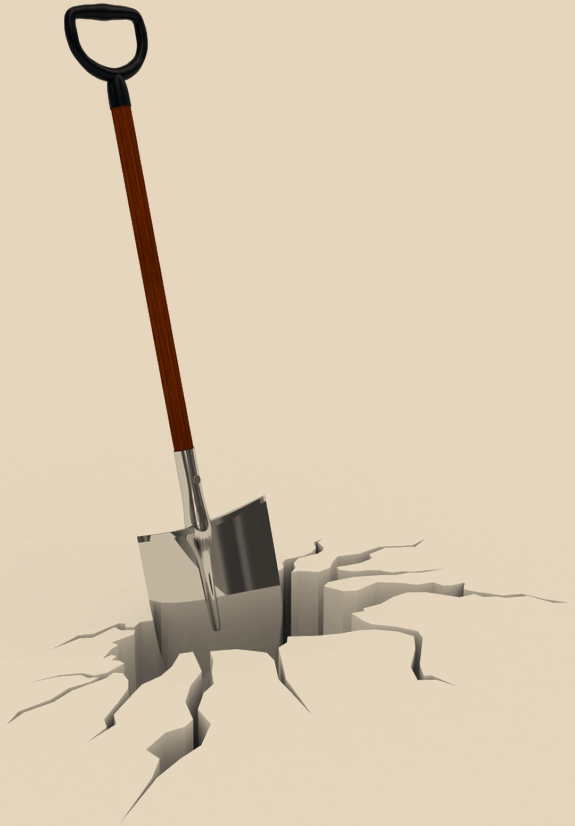
overriding CORE functions

Install a resub on any CORE function with a prototype

Writing tests is work.

Refactor your tests aggressively.

- CPAN is great, and almost solves all your problems...
- ...but ultimately your problems are unique like you, snowflake...
- ...so you should feel empowered to write out your own testing library.



Thanks

belden@shutterstock.com

(Hey, we're hiring)