Test::Class::Moose

Write your tests as Moose classes

Julien Fiegehenn (simbabque)

Why test?

Large applications have complicated test suites

Long .t files are horrible

What can we do about that?

Test code is production code!

Production code has classes

```
package Person;
use Moose;

has name => (
    is       => 'ro',
    required => 1,
);

has age => (
    is       => 'ro',
    isa      => 'Int',
    required => 1,
    writer   => '_write_age',
);

sub birthday {
    my ($self) = @_;

    $self->_write_age( $self->age + 1 );
}
```

The tests can have classes too!

```
package TestsFor::Person;
use Test::Class::Moose;
use Person;

sub test_birthday {
    my ($test) = @_;

    my $bob = Person->new( name => 'Bob', age => '20' );
    is $bob->age, 20, 'Bob starts with the right age';
    $bob->birthday;
    is $bob->age, 21, 'Bob is a year older after his birthday';
}

1;
```

We need a way to load it.

```
use Test::Class::Moose::Load 't/lib';
use Test::Class::Moose::Runner;
Test::Class::Moose::Runner->new->runtests;
```

What does this look like now?

```
julien@horst:~/code/private/talks $ tree test-class-moose
test-class-moose
├── lib
│   └── Person.pm
└── t
    ├── lib
    │   └── TestsFor
    │       └── Person.pm
    └── tests.t
```

Let's run it!

```
julien@horst:~/code/private/talks/test-class-moose $ prove -lv t
t/tests.t ..
1..1
#
# Running tests for TestsFor::Person
#
    # Subtest: TestsFor::Person
    1..1
    # TestsFor::Person->test_birthday()
        # Subtest: test_birthday
        ok 1 - Bob starts with the right age
        ok 2 - Bob is a year older after his birthday
        1..2
    ok 1 - test_birthday
ok 1 - TestsFor::Person
ok
All tests successful.
Files=1, Tests=1,
1 wallclock secs ( 0.02 usr  0.00 sys +  0.33 cusr  0.03 csys =  0.38 CP
Result: PASS
```

Let's add another test

```
sub test_name {
    my ($test) = @_;

    dies_ok { my $bob = Person->new( name => 'Bob' ) }
        'Creating Bob without an age blows up';
}
```

```
#
# Running tests for TestsFor::Person
#
    # Subtest: TestsFor::Person
    1..2
    # TestsFor::Person->test_birthday()
        # Subtest: test_birthday
        ok 1 - Bob starts with the right age
        ok 2 - Bob is a year older after his birthday
        1..2
    ok 1 - test_birthday
    # TestsFor::Person->test_name()
        # Subtest: test_name
        ok 1 - Creating Bob without an age blows up
        1..1
    ok 2 - test_name
ok 1 - TestsFor::Person
```

Overview

- one test class for each class
- one `test_` sub for each test case or method

What else can we do?

The tests are Moose classes, so they can have attributes.

```
package TestsFor::Foo;
use Test::Class::Moose;
use DBI;

has dbh => (
  is => 'ro',
  default => sub {
    DBI->connect( '...' );
  }
);

sub test_frobnicate {
  my ($test) = @_;

  my $sth = $test->dbh->prepare( '...' );
}
```

Tag tests to organize them into logical unit

```
sub test_save_poll_data : Tags(api network) {
    ...
}
```

And then filter by tags in the runner config

```
Test::Class::Moose::Runner->new(
    include_tags => [qw/api database/],
    exclude_tags => 'deprecated',
)->runtests;
```

Only run specific test classes.

```
Test::Class::Moose::Runner->new(
    test_classes => [qw/TestsFor::Specific::Class/],
)->runtests;
```

Test::Class::Moose gives you Test::Most

- strict
- warnings
- Test::More
- Test::Exception
- Test::Difference
- Test::Deep
- Test::Warn

Remember it's Moose!

- subclass your tests
- use roles to load test features (like a mocked database)

Test::Class::Moose

written by Ovid

current maintainer Dave Rolsky

https://metacpan.org/pod/Test::Class::Moose