OSADO Unit testing

[an introduction]

Andrea Manzini

andrea.manzini@suse.com

- What ? 🤔

- Why ? 🤌

- Who ? 💁 (you!)

- How ? 👀

I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO, I DO IT IN PRODUCTION

What's unit testing ?

- Unit testing, a.k.a. component or module testing, is a form of software testing by which isolated portions of source code is tested to validate expected behavior.
- We do a lot of testing in our daily job, but we (almost) never test our code.
- A typical unit testing example is to ensure a given function returns the expected value or errors given some predefined inputs.

Why Unit Tests ?

- they can catch early-stage bugs
- They function as regression validator
- Run much faster and require less resource than Integration Tests
- Easy to run automatically
- The act as "live documentation"
- Great way to "explore" and refactor a unknown codebase

An example

```perl
sub get_java_versions {
    # on newer version we need legacy module for openjdk 11, but is not available
    # on SLERT/SLED, can't test openjdk 11. On 15-SP7 17 is also in legacy module
    return '21' if (is_rt || is_sled) && is_sle('>=15-SP7');
    return '11 17 21' if (is_sle '>=15-SP6');
    return '17 21' if ((is_rt || is_sled) && is_sle('>=15-SP6'));
    return '11 17';
}

sub run {
    my $self = @_;

    my @java_versions = split(' ', get_java_versions);
    ...
```

from `openjdk_fips.pm`

1. move the function to a separate library `lib/security/openjdk_utils.pm`

```perl
package security::openjdk_utils;

use strict;
use warnings;
use version_utils qw(is_sle is_sled is_rt);

use base 'Exporter';

our @EXPORT = qw(get_java_versions);

sub get_java_versions {
    # on newer version we need legacy module for openjdk 11, but is not available
    # on SLERT/SLED, can't test openjdk 11. On 15-SP7 17 is also in legacy module
    return '21' if (is_rt || is_sled) && is_sle('>=15-SP7');
    return '11 17 21' if (is_sle '>=15-SP6');
    return '17 21' if ((is_rt || is_sled) && is_sle('>=15-SP6'));
    return '11 17';
}

1;
```

2. import the library in the test module and everything keeps running as before

```
# openjdk_fips.pm
use security::openjdk_utils qw(get_java_versions);

sub run {
    my $self = @_;

    my @java_versions = split(' ', get_java_versions);
    ...
```

3. create first unit test `t/36_openjdk.t`

```
use strict;
use warnings;
use Test::More;
use Test::Warnings;
use Test::MockModule;
use List::Util qw(any none);
use testapi;
use security::openjdk_utils 'get_java_versions';

subtest '[dummy]' => sub {
    my $dummy = 3 + 3;
    ok($dummy == 6, 'check that 3+3 == 6 (setup works)');
};
```

4. run it from osado:

```
$ 'make prepare'
$ sudo zypper in perl-App-cpanminus
$ export PERL5LIB=~/perl5:~/Work/os-autoinst:~/Work/os-autoinst-distri-opensuse/lib
$ export PERL5OPT=-MCarp::Always

# run a sample existing test, to check it all works:

$ time prove --time --verbose -l  t/33_qam.t

# run our dummy test

$ time prove --time --verbose -l  t/36_openjdk.t
```

5. improve the unit tests

```perl
subtest '[default behavior]' => sub {
  my $java_versions = get_java_versions();
  ok($java_versions eq '11 17',
    'check that 11 and 17 are returned by default');
};
```

Dependencies issue!

The `get_java_versions` function will indirectly call `is_rt()`, `is_sled()` and so on, and these functions will return significant values only when the code is run inside `openQA`, depending on the exact product version under testing.

How do I create a *fake* `is_sle('16')` and so on ?

Mocking (basic)

```
subtest '[test for any SLE, no RT, no SLED]' => sub {
  my $mocked_module = Test::MockModule->new('security::openjdk_utils');
  $mocked_module->redefine(
    'is_sle' => sub { return 1; },
    'is_rt' => sub { return 0; },
    'is_sled' => sub { return 0; });
  my $java_versions = get_java_versions();
  note "Java versions: $java_versions\n";
  ok($java_versions eq '11 17 21',
   'check that 11 17 21 are returned by default SLE');
};
```

reference: https://metacpan.org/pod/Test::MockModule

Mocking (with forced args)

```
subtest '[test for any SLE, no RT, no SLED]' => sub {
  my $mocked_module = Test::MockModule->new('security::openjdk_utils');
  $mocked_module->redefine(
    'is_sle' => sub { return ($_[0] eq '16+') ? 1 : 0 },  # <-
    'is_rt' => sub { return 0; },
    'is_sled' => sub { return 0; });
  my $java_versions = get_java_versions();
  note "Java versions: $java_versions\n";
  ok($java_versions eq '11 17 21',
   'check that 11 17 21 are returned by default SLE');
};
```

Mocking (and testing args)

example with the classic openQA test module that just does a bunch of `assert_script_run`

```perl
package security::openssl;

use strict;
use warnings;

use base 'Exporter';

our @EXPORT = qw(do_ssl_testing);

sub do_ssl_testing {
    assert_script_run 'systemctl ...'
    assert_script_run 'openssl ...'
    ...
}
```

Mocking (and testing args)

we can mock `assert_script_run` and check the command called are the one expected, and return the value we want

```
subtest '[test for any SLE, no RT, no SLED]' => sub {
  my $func = Test::MockModule->new('security::openssl', no_auto => 1);
  my @calls;
  $func->redefine(assert_script_run =>
    sub { push @calls, ['', $_[0]]; return; });

  $func->redefine(script_output =>
    sub { return 'the very long string command output'; }););

  ok(($#calls > 0), "There are some command called");
  ok((any { /systemctl / } @cmds),
    'executed commands contains systemctl');
}
```

Further references

- OSADO unit tests
- Tutorial on UT in Perl
- EXP Chapter 13: Unit Testing

OSADO Unit testing

[an introduction]

Thanks for watching!

Questions ?

andrea.manzini@suse.com