

Модульные тесты

нахрена?

Преимущества использования

Поощрение изменений

Я могу спокойно заниматься рефакторингом и быть уверенным, что все будет работать как и прежде

Возможность тестирования приложения по частям

Мне не нужно собирать полностью рабочую систему для проверки работоспособности одного компонента

Преимущества использования

Грамотный интерфейс модулей

Когда я начинаю создавать тест до написания основного кода, я могу быть уверен, что модулем будет удобно пользоваться

```
my $object = local::Factory::Some::DB::Object->read({id  
=> 5})->[0];  
local::Factory::Some::DB::Object->update({ id => $var-  
>id, name => 'new_name'})
```

Такого кода у вас точно не будет!

Преимущества использования

Уверенность в себе

Я могу быть уверенным, что мой старый код, которым уже никто не занимается работает как и прежде

Дадим отдохнуть QA

Мы будем отлавливать множество багов ещё на этапе разработки. До QA дойдут только лучшие из наших багов.

Чем нам поможет Perl?

Test::More

расскажу о функциях...

```
use_ok('Some::Module');  
ok( 1, 'all is ok' );  
ok( 0, 'error' )  
is( 'some data', 'some data', 'string are equal' );  
like( 'some data', qr/some/, 'this will fail' );  
is_deeply( { a=>1, b => 2 },{ a => 1, b => 2});  
pass;  
fail;
```

```
perldoc Test::More
```

Пример теста

```
#!/usr/bin/env perl
use Test::More tests => 10;
ok( 1 < 2, '1 < 2' );
SKIP: {
    skip 'We need to have 5.10 to use this', 1 if $] < 5.010;
    use feature qw( switch say );
    ok( say('some output'), 'say' );
};
isnt( $^O, 'windows' );
foreach ( 4..10 ) {
    rand(2) ? pass : fail;
}
```

Некоторые полезные модули

1. `Test::Pod::Coverage` - проходит только если для каждого метода описана Pod - документация
2. `Test::Exception` - если у вас используются исключения
3. `Devel::Cover` - показывает, какие строки вашего кода выполняются при тестировании (http://gugu.static.dev.rambler.ru/cover_db/coverage.html)
4. `Test::Perl::Critic` - находит типичные ошибки программистов

А что с Web-ом?

Как нам тестировать сайт?

WWW::Mechanize

Эмуляция браузера.

- можем ходить по ссылкам
- заполнять и отправлять формы

Test::WWW::Mechanize

Класс для тестирования веб-приложений. Содержит в себе кучу полезных методов.

На нем мы остановимся поподробней.

Методы Test::WWW::Mechanize

- `get_ok('path')` - успех, если GET-запрос к `path` завершился успешно
- `submit_form_ok({ form_number => 1, fields=> $data})` - сабмитит первую форму на странице. успех, если запрос завершился успешно
- `title_is('some title')` - заголовок страницы должен быть "some title"
- `content_contains('some content')`
- `follow_link_ok(text => 'Удалить')` - переходит по ссылке с ЭТИМ ИМЕНЕМ

Маленький пример

```
#!/usr/bin/env perl
use Test::More tests => 5;
use Test::WWW::Mechanize;

my $mech = Test::WWW::Mechanize->new;
$mech->get_ok( $page );
$mech->base_is( 'http://petdance.com/', 'Proper <BASE
HREF>' );
$mech->title_is( "Invoice Status", "Make sure we're on the
invoice page" );
$mech->content_contains( "Andy Lester", "My name
somewhere" );
$mech->content_like( qr/(cpan|perl)\.org/, "Link to perl.org or
CPAN" );
```

И напоследок

Никакие unit-тесты не заменят вам:

- отдел тестирования
- качественный код
- и мозги

Они все лишь немного упростят вам жизнь!