Test Driven Development

Software garatzeko modu bat, metodologia bat

Testa prestatu eta gero hori pasatzeko programatu

Test Driven Development

GORRIA => BERDEA => REFAKTORIZATU

Terminoak

- Unit Test

  funtzionalitate txiki-jakin batentzako testa, espero duguna egiten du maila baxuan: etiketak parseatzea....

  - **Mock** (adibidez, registratzean bidali emaila (mailer), db eguneratu (repository)...). Ez dira exekutatzen, “mockeatu” egiten dira.
  - **Stub** (emaitzak-*behaviour* simulatu mockari)
  - **Dummies**: berdin zaigunean objektuak zer egiten duen. Funtzio batek ez du behar auth, klasean ordea badago... return null;

- Integration Test: aplikazioaren atal ezberdinak testeatzeko, ruta+gorde erabiltzailea+...(db,I/O...) Ez da mokeatzen, ez stubs...
  - **codeception,behat,phpunit,behat.**..bdd?acceptance?

Proiektuaren garapena

55% programatzen

20% integratzen (+fixing)

25% testeatzen (+fixing)

Hori proiektua kudeatzen duena jatorra bada, eta denbora baldin badago.

Garapena TDD erabilita

70% programatzen

10% integratzen

15% testeatzen

Eta emaitza, osotara ordu gutxiago

BUG-ak lehengo aurkituko dira, eta errazago debugeatuko dira ere. Testen bidez zatitxoak testeatzen ditugulako.

Garapena TDD erabilita

Proiektu txikietan gutxiago nabarituko da

Handietan eta etengabe aldatu behar direnetan asko!

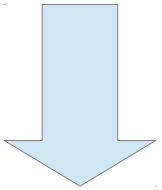
Abantailak

Kode hobea, garbiagoa (SOLID...)

Mantenigarriagoa

Refaktorizazioa hobe

Aldaketak egiteko hobe



PRODUKTIBOAGOAK

LASAITASUNA

...

Lengoaiak eta tresnak

Edozein lengoaietan aplika daiteke TDD

PHP: phpUnit, phpspec...

Java: JUnit

JS: Qunit,Jasmine,protractor...

objective c, **swift**, **python**...

Adibideak nire proiektuetan

- Katak egin, praktikatu, praktikatu...
- Kostatzen da hastea...
- Baina adibideak: P3,P6 egutegiak, date validations, integrazioak..
- Aplikazio handi pare batekin (15 hilabeteko garapena... testak hasieran, gero ez. Azkenean kode zakarra, debugeatzeko lanak... orain testak egiten!)

PHPUnit

composer.json (http://composer.json)

```
{
 "require": {
  "phpunit/phpunit": "~4.0"
 },
 "autoload": {
  "psr-4": {
   "Biltzarra\\": "src/Biltzarra"
  }
 }
}
```

PHPUnit

```php
use Biltzarra\Calculator;

class CalculatorTest extends \PHPUnit_Framework_TestCase
{


    public function test_class_exists()
    {

        $calculator = new Calculator()

        return $this->assertInstanceOf('Biltzarra\Calculator',$calculator);
    }
```

PHPUnit

```php
public function test_empty_string_returns_0()
{
    $result = $this->calculator->calculate('');

    return $this->assertEquals(0,$result);
}
```

PHPSpec

```
{
 "autoload": {
  "psr-0": {
   "": "src/"
  }
 },
 "require-dev": {
  "phpspec/phpspec": "^3.1"
 }
}
```

PHPSpec

Phpspec describe StringCalculator

phpspec run

phpscpen run -fpretty

PHPSpec

```
class StringCalculatorSpec extends ObjectBehavior
{
    function it_is_initializable()
    {
        $this->shouldHaveType(StringCalculator::class);
    }


    function it_returns_0_with_empty_string()
    {
        $this->add('')->shouldEqual(0);
    }


function it_returns_the_sum_of_two_numbers()
{
    $this->add('1,2')->shouldEqual(3);
}
```