Tests unitaires

Présentation

“Tester… ce n’est pas douter !”

@laryakan

A quoi ça sert les tests unitaires ?

À simuler un comportement afin de tester unitairement !

Tester unitairement, c'est s'assurer du fonctionnement d'une partie précise de son programme (unité), indépendamment d'autres composants/unités. C'est pourquoi on simule le reste (via des doublures).

Un test doit aller le plus vite possible. Il doit s'exécuter rapidement et être compréhensible dans sa lecture. Il a vocation à être exécuté de manière récurrente en mettant explicitement en relief le moindre dysfonctionnement.

```php
<?php
namespace App\Test\Model;
use App\Model\Deck;
use PHPUnit\Framework\TestCase;

class DeckTest extends TestCase
{
	/** @test */
	public function testConstruct()
	{
		$deck = new Deck(30);
		$this->assertInstanceOf('App\Model\Deck', $deck);
		$this->assertEquals(count($deck->cards), 30);
	}
}
```

Les tests unitaires, une pratique agile

Décrite dans la méthode “Extreme Programming” (1999), les tests unitaires font pleinement partie des pratiques agiles.

Ils servent à valider techniquement les développements et sont souvent un composant de la “Definition of Done” dans le framework Scrum.

Ceci permet de valider les développements d’un Récit Utilisateur (User Story).

```php
<?php
namespace App\Test\Model;
use App\Model\Deck;
use PHPUnit\Framework\TestCase;

class DeckTest extends TestCase
{
	/** @test */
	public function testConstruct()
	{
		$deck = new Deck(30);
		$this->assertInstanceOf('App\Model\Deck', $deck);
		$this->assertEquals(count($deck->cards), 30);
	}

}
```

PHPUnit

Afin de tester unitairement, plutôt que de tester des résultats à coup de “if()” ou de “switch()”, il existe des frameworks et librairies de test.

Qu’il s’agisse libcbdd en C, Bandit en C++, mocha en Javascript, JUnit en Java, la grande majorité des langages de programmation, interprétés ou non, disposent de framework de test.

PHP ne fait pas exception avec PHPUnit.

```php
<?php
namespace App\Test\Model;
use App\Model\Deck;
use PHPUnit\Framework\TestCase;

class DeckTest extends TestCase
{
	/** @test */
	public function testConstruct()
	{
		$deck = new Deck(30);
		$this->assertInstanceOf('App\Model\Deck', $deck);
		$this->assertEquals(count($deck->cards), 30);
	}

}
```

On s’y prend comment ?

Concrètement, un test unitaire se résume à vérifier que le résultat correspond à une attente (“expectation”) via une assertion.

Pour cela, et afin de s’assurer du caractère “unitaire” du test, il faut contextualiser son exécution.

Pour cela, un test unitaire doit se limiter à tester une et une seule fonction, relativement à une liste de paramètres. Nous en vérifierons le résultat, la production ou la sortie.

assertion
nom féminin
(latin adsertio, de adserere, affirmer)

Définitions

1. Proposition, de forme affirmative ou négative, qu'on avance et qu'on donne comme vraie.
2. Affirmation catégorique de quelque chose qu'il n'est pas possible de vérifier.
3. Statut d'une phrase dans laquelle le sujet parlant énonce une vérité, déclare un fait (par opposition à l'interrogation, à l'exclamation, à l'injonction).
4. Opération qui consiste à poser la vérité d'une proposition et qui est généralement symbolisée par le signe placé devant elle ; cette proposition.

```
public function testAssertTrue()
{
    $this->assertTrue(true);
}
```

Comment contextualiser une exécution de test ?

Afin de contextualiser l'exécution d’un test, il faut prendre soin d’initialiser de manière arbitraire son environnement d'exécution.

En POO, cela consiste à positionner toutes les propriétés de sa classe d'exécution, et à en simuler toutes les dépendances.

Ceci signifie que vos développements doivent êtres génériques au point de pouvoir en maîtriser les dépendances de l’extérieur, par exemple via la rédaction de méthodes publiques de type “set”.

Controller

```
[...]
public function setPlayers(Player $playerOne, Player $playerTwo, $deck
= null)
{
        [...]
}
[...]
```

Test

```
[...]
$playerSpyOne = new PlayerSpy('PlayerSpyOne');
$playerSpyTwo = new PlayerSpy('PlayerSpyTwo');
$game->setPlayers($playerSpyOne, $playerSpyTwo, $deck);
[...]
```

Comment maîtriser le contexte ?

Afin d'isoler notre fonction de son environnement, il est nécessaire de le simuler.

Puisque nous programmons l'environnement, nous en connaissons le comportement. La partie variable reste donc la fonction que nous voulons tester.

Afin de simuler l'environnement, nous allons avoir recours à des ***doublures de test***.

```
namespace App\Test\Spy;

use App\Model\Player as OriginalPlayer;
use App\Model\Deck;

class Player extends OriginalPlayer
{
        [...]
}
```

La suite

Cette présentation vient en introduction du Coding Dojo (Kata) sur

Les Doublures de Test

https://drive.google.com/open?id=1QEedSEPIByz4X4CCTdyRUiuLq5Cq-VRu3CvdXedv_Cw