# Test de contrat : Ne pas découvrir le pot aux roses en production

*Utiliser des doublures, et plus particulièrement des fakes, permet de mieux contrôler nos tests et d'accélérer leur exécution. Mais comment peut-on s'assurer que les doublures reproduisent fidèlement le comportement de ce qu'elles remplacent ?*

## Deux implémentations qui diffèrent

À la fin du chapitre précédent, nous avions deux implémentations de l'interface `BurritoDuJourRepository` :

```php
interface BurritoDuJourRepository
{
    public function ajouteBurritoDuJour(BurritoDuJour $burritoDuJour): void;

    public function burritoPourLe(string $date): BurritoDuJour;
}
```

La première s'appuie sur une base de données et est utilisée en production :

```php
final class BurritoDuJourRepositoryEnBaseDeDonnées implements BurritoDuJourRepository
{

    public function __construct(
        private readonly \PDO $pdo,

    ) {
    }

    public function ajouteBurritoDuJour(BurritoDuJour $burritoDuJour): void
    {
        $stmt = $this->pdo->prepare('INSERT INTO burrito_du_jour (date, nom) VALUES (:date,
:nom)');
        $stmt->execute([
            'nom' => $burritoDuJour->nom(),
            'date' => $burritoDuJour->date(),
        ]);
    }

    public function burritoPourLe(string $date): BurritoDuJour
    {
        $stmt = $this->pdo->prepare('SELECT nom, date FROM burrito_du_jour WHERE date = :date');
        $stmt->execute(['date' => $date]);
        $data = $stmt->fetch();

        return new BurritoDuJour(
            $data['date'],
            $data['nom']
        );
    }
}
```

Et la seconde, utilisée dans les tests, stocke les `BurritoDuJour` en mémoire :

```php
final class BurritoDuJourRepositoryEnMémoire implements BurritoDuJourRepository
{

    private array $burritosParDate;

    public function ajouteBurritoDuJour(BurritoDuJour $burritoDuJour): void
    {
        $this->burritosParDate[$burritoDuJour->date()] = $burritoDuJour;
    }

    public function burritoPourLe(string $date): BurritoDuJour
    {
        return $this->burritosParDate[$date];
    }
}
```

Nous avions également un test pour nous assurer que l'implémentation de production fonctionnait comme prévu.

```php
class BurritoDuJourRepositoryEnBaseDeDonnéesTest extends TestCase
{
    /**
     * @test
     */
    public function récupère_un_burrito_du_jour_à_une_date(): void
    {
        $connexion = DatabaseConnection::getInstance();
        $repository = new BurritoDuJourRepositoryEnBaseDeDonnées($connexion);

        $burritoDuJour = new BurritoDuJour('14-06-2036', 'Kebab');

        $repository->ajouteBurritoDuJour($burritoDuJour);

        $this->assertEquals($burritoDuJour, $repository->burritoPourLe('14-06-2036'));
    }
}
```

Bien que les deux implémentations respectent l'interface, cela ne garantit pas qu'elles aient le même comportement. Par exemple, l'implémentation pour les tests pourrait retourner le `BurritoDuJour` de la veille de la date demandée, tandis que celle s'appuyant sur la base de données, utilisée en production, retourne le bon `BurritoDuJour`.

De plus, il se pourrait que l'on souhaite que le repository lance une exception si aucun `BurritoDuJour` n'est trouvé pour la date demandée.

Pour implémenter ce comportement, ajoutons un test pour notre `BurritoDuJourRepositoryEnBaseDeDonnées` :

```php
    /**
     * @test
     */
    public function jette_une_exception_si_aucun_burrito_nest_trouvé_pour_la_date(): void
    {
        $connexion = DatabaseConnection::getInstance();
        $repository = new BurritoDuJourRepositoryEnBaseDeDonnées($connexion); ②

        $this->expectExceptionMessage("Aucun burrito trouvé pour le 16-09-2045."); ①

        $repository->burritoPourLe("16-09-2045");
    }
```

Dans ce test, nous instancions un `BurritoDuJourRepositoryEnBaseDeDonnées` ❶ sans y ajouter aucun `BurritoDuJour`. Nous attendons qu'une exception soit levée ❷, puis tentons de récupérer un burrito pour une date quelconque.

Ce test nous pousse à modifier le `BurritoDuJourRepositoryEnBaseDeDonnées` pour qu'il lance une exception :

```php
    public function burritoPourLe(string $date): BurritoDuJour
    {
        $stmt = $this->pdo->prepare('SELECT nom, date FROM burrito_du_jour WHERE date = :date');
        $stmt->execute(['date' => $date]);
        $data = $stmt->fetch();

        if ($data === false) {
            throw new \Exception(sprintf("Aucun burrito trouvé pour le %s.", $date));
        }

        return new BurritoDuJour(
            $data['date'],
            $data['nom']
        );
    }
```

Malheureusement, le `BurritoDuJourEnMémoire` ne lance pas d'exception quand aucun burrito n'est trouvé.

Pour remédier à ce problème, nous allons utiliser un test de contrat. Ce test va s'assurer que différentes implémentations réagissent de la même manière, répondant ainsi au même contrat.

## Test de contrat

Pour mettre en place ce test de contrat, nous allons nous baser sur les tests vérifiant le bon fonctionnement du `BurritoDuJourRepositoryEnBaseDeDonnées`, qui est notre implémentation de production.

Nous créons une classe de test abstraite, `BurritoDuJourRepositoryTest`, qui va spécifier le comportement attendu pour toutes les implémentations :

```php
abstract class BurritoDuJourRepositoryTest extends TestCase ①
{

    abstract protected function repository(): BurritoDuJourRepository; ②

    /**
     * @test
     */
    public function jette_une_exception_si_aucun_burrito_nest_trouvé_pour_la_date(): void
    {
        $repository = $this->repository(); ③

        $this->expectExceptionMessage("Aucun burrito trouvé pour le 16-09-2045.");

        $repository->burritoPourLe("16-09-2045");
    }

    /**
     * @test
     */
    public function récupère_un_burrito_du_jour_à_une_date(): void
    {
        $repository = $this->repository(); ④

        $burritoDuJour = new BurritoDuJour('14-06-2036', 'Kebab');

        $repository->ajouteBurritoDuJour($burritoDuJour);

        $this->assertEquals($burritoDuJour, $repository->burritoPourLe('14-06-2036'));
    }
}
```

Cette classe est déclarée abstraite, avec `abstract` ❶, pour indiquer qu'elle ne peut pas être instanciée directement. Elle exige également que toutes ses sous-classes fournissent une méthode `repository` retournant un `BurritoDuJourRepository` ❷.

Cette méthode est utilisée dans les tests ❸ ❹ pour obtenir un repository et effectuer des vérifications sur son comportement.

La classe de test pour le `BurritoDuJourRepositoryEnBaseDeDonnées` s'écrit alors comme suit :

```php
class BurritoDuJourRepositoryEnBaseDeDonnéesTest extends BurritoDuJourRepositoryTest ①
{
    protected function repository(): BurritoDuJourRepository ②
    {
        $connexion = DatabaseConnection::getInstance();

        return new BurritoDuJourRepositoryEnBaseDeDonnées($connexion);
    }
}
```

Cette classe de test hérite de `BurritoDuJourRepositoryTest` ❶ et implémente la méthode `repository`, responsable de construire le `BurritoDuJourRepositoryEnBaseDeDonnées` utilisé dans les tests ❷.

Pour tester le repository en mémoire, il suffit de créer une classe de test étendant `BurritoDuJourRepositoryTest` ❶ et fournissant une instance de `BurritoDuJourEnMémoire` ❷ :

```php
class BurritoDuJourRepositoryEnMémoireTest extends BurritoDuJourRepositoryTest ①
{
    protected function repository(): BurritoDuJourRepository
    {
        return new BurritoDuJourRepositoryEnMémoire();
    }
}
```

Lorsque nous exécutons les tests, l'un d'eux échoue, indiquant que le repository en mémoire ne lance pas d'exception lorsqu'aucun burrito n'est trouvé. Cela nous incite à ajuster l'implémentation du `BurritoDuJourEnMémoire` pour aligner son comportement avec celui de l'implémentation de production :

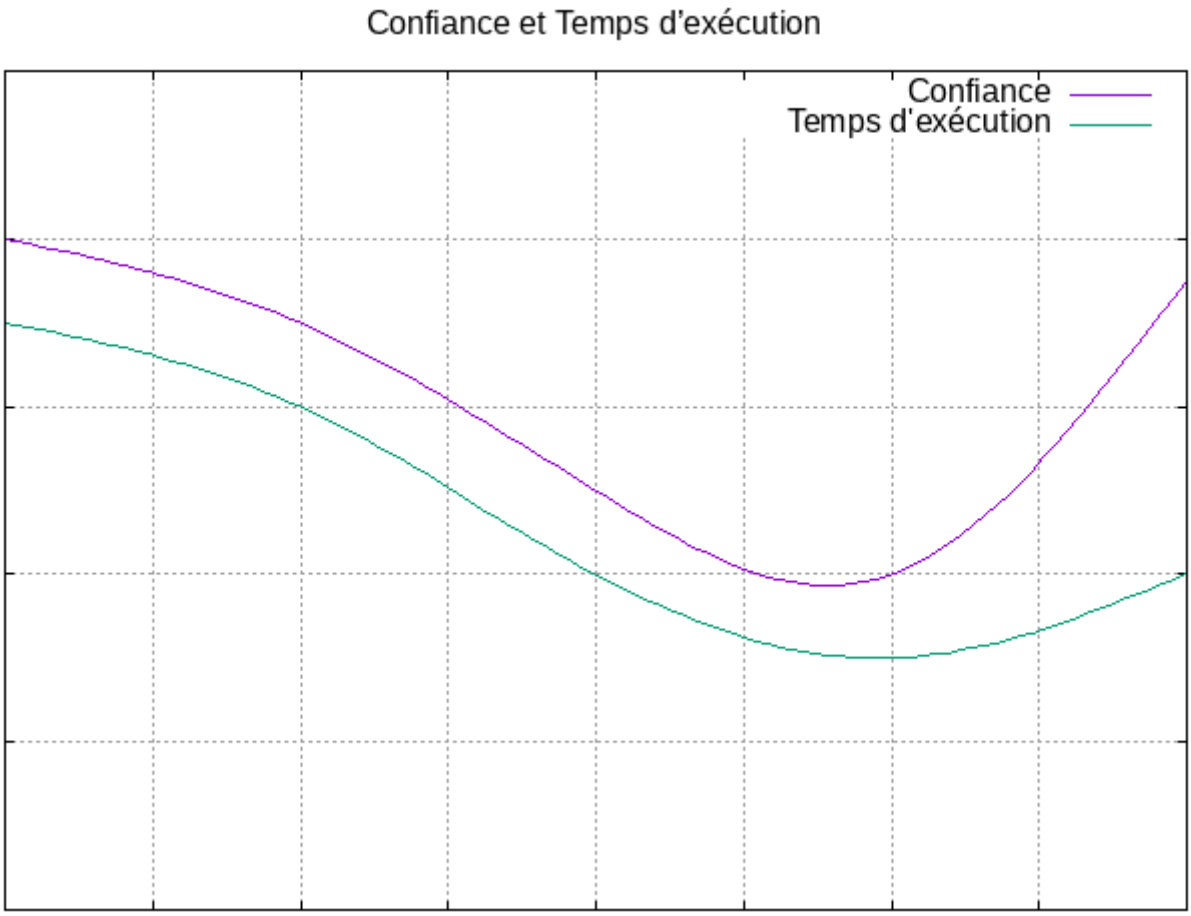
```php
    public function burritoPourLe(string $date): BurritoDuJour
    {
        return $this->burritosParDate[$date] ?? throw new \Exception(
            sprintf("Aucun burrito trouvé pour le %s.", $date)
        );
    }
```

Grâce au test de contrat, nous pouvons maintenant nous assurer que toutes les implémentations réagissent de manière cohérente. Cela réduit le risque de découvrir en production des différences de comportement entre notre doublure et l'implémentation de production.

Les tests de contrat permettent de vérifier que différentes implémentations, qu'elles soient utilisées en production ou uniquement dans les tests, se comportent de la même manière.

## Aligner la confiance et la vitesse

En utilisant des doublures, nous perdons progressivement en confiance. Ce qui nous offrirait le plus de confiance serait d'utiliser systématiquement les implémentations de production, garantissant ainsi de ne pas se baser sur des doublures au comportement différent. Cependant, l'utilisation des implémentations de production peut parfois ralentir significativement les tests.




Les tests de contrat offrent un compromis intéressant, car ils augmentent notre confiance dans le fait que nos doublures se comportent comme les véritables implémentations. Nous pouvons ainsi réduire le temps d'exécution des tests en utilisant des doublures, tout en limitant l'utilisation des implémentations plus lentes aux tests de contrat essentiels.

## Variation avec un Data Provider

Il est possible de mettre en place des tests de contrat à l'aide d'un data provider. Pour cela, il faut définir chaque test pour qu'il accepte en argument l'implémentation à tester et s'assurer que le data provider fournisse les différentes implémentations.

Voici ce que donnerait cette variation dans l'exemple de nos tests de `BurritoDuJourRepository` :

```php
class BurritoDuJourRepositoryAlternativeTest extends TestCase
{
    /**
     * @test
     * @dataProvider repositories
     */
    public function jette_une_exception_si_aucun_burrito_nest_trouvé_pour_la_date
(BurritoDuJourRepository $repository
    ): void {
        $this->expectExceptionMessage("Aucun burrito trouvé pour le 16-09-2045.");

        $repository->burritoPourLe("16-09-2045");
    }

    /**
     * @test
     * @dataProvider repositories
     */
    public function récupère_un_burrito_du_jour_à_une_date(BurritoDuJourRepository $repository):
void
    {
        $burritoDuJour = new BurritoDuJour('14-06-2036', 'Kebab');

        $repository->ajouteBurritoDuJour($burritoDuJour);

        $this->assertEquals($burritoDuJour, $repository->burritoPourLe('14-06-2036'));
    }

    public function repositories()
    {
        yield [new BurritoDuJourRepositoryEnMémoire()];

        $connexion = DatabaseConnection::getInstance();
        yield [new BurritoDuJourRepositoryEnBaseDeDonnées($connexion)];
    }
}
```