Test Driven Development

Luca Bovino

TRIANON
Compensation
& Benefits Management

Overview

- Tester… mais quoi?
- Les trois lois du TDD
- Les bonnes habitudes
- Le TPP

Tester… mais quoi?

~~Cogito~~ Testor Ergo Sum

NOUS TESTONS LE **COMPORTEMENT**, PAS L'IMPLÉMENTATION

```
public class CalculatorShould {
   [Fact]
   public void SumTwoIntegers() {
      var calculator = new Calculator();
      var sum = calculator.add(2, 1);

      Assert.Equals(sum, 3);
   }
}
```

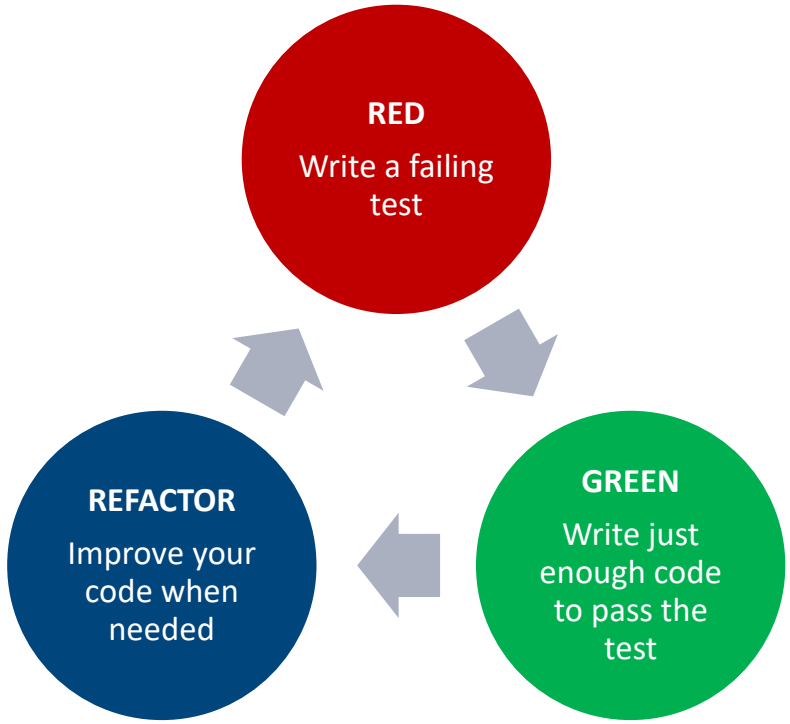
```
public class Calculator {
   public int add(int a, int b) {
      return a + b;
   }
}
```

```
public class Calculator {
   private AdderFactory adderFactory;

   public Calculator(AdderFactor adderFactory)
   {
      this.adderFactory = adderFactory;
   }

   public int add(int a, int b)
   {
      Adder adder = adderFactory.createAdder();
      ReturnValue returnValue = adder.compute(new Number(a), new Number(b));
      return returnValue.convertToInteger();
   }
}
```

```
public class CalculatorShould {
   [Fact]
   public void SumTwoIntegers() {
      var calculator = new Calculator();
      var sum = calculator.add(2, 1);

      Assert.Equals(sum, 3);
   }
}
```

Les trois lois du TDD

Improvise, Adapt and Overcome!



1. Commencez pas écrire **un seul** test avec **une seule** théorie
2. Implémentez le minimum de code afin de passer **cette** théorie
3. Enrichissez vos test avec une nouvelle théorie ou un nouveau test
4. *Répétez les points 2 & 3*
5. Votre code de production est là avec tous les filets de sécurité qui vont avec !

REFACTOR : utiliser la règle de trois -> refactorer uniquement lorsqu'on a 3 répétitions et pas avant

Les bonnes habitudes

“N'importe quel imbécile peut écrire du code qu'un ordinateur peut comprendre.

Les bons programmeurs écrivent du code que les humains peuvent comprendre”.

En écrivant des test

Les tests doivent expliquer à **l'humain** le fonctionnement du code de production.

```
public class CalculatorShould {
   [Fact]
   public void SumTwoIntegers() {
      var calculator = new Calculator(); // GIVEN
      var sum = calculator.add(2, 1);    // WHEN

      Assert.Equals(sum, 3);             // THEN - Commencer ici!
   }
}
```

Un test doit être

Rapide : chaque seconde compte

Isolé : pas de dépendances entre tests

Répétable : toujours le même résultat

Auto-validant : rouge ou vert

En récupérant le code des autres

1ère chose à faire -> **Lire et lancer** les tests

Le TPP

Au fur et à mesure que les tests deviennent plus spécifiques, le code devient plus générique

Le **T**ransformation **P**riority **P**remise défini une liste de transformations du code classées par complexité.

Les transformations **en haut** de la liste **doivent être préférées** à celles plus bas.

Cela permet de garder le code simple et compréhensible.

Exemple d'implémentation pas-à-pas : https://blog.cleancoder.com/uncle-bob/2013/05/27/TheTransformationPriorityPremise.html

Le TPP

- **({}—>nil/null)** no code at all->code that employs nil/null
- **(nil/null->constant)**
- **(constant->constant+)** a simple constant to a more complex constant
- **(constant->scalar)** replacing a constant with a variable or an argument
- **(statement->statements)** adding more unconditional statements.
- **(unconditional->if)** splitting the execution path
- **(scalar->array)**
- **(array->container)**
- **(statement->recursion)**
- **(if->while)**
- **(expression->function)** replacing an expression with a function or algorithm
- **(variable->assignment)** replacing the value of a variable.

Le TPP

Exemple : Kata "Word Wrap"

```
@Test
public void WrapNullReturnEmptyString()
{
  assertThat(wrap(null, 10), is(""));
}
```

({}–>nil/null)

```
public static String wrap(String s, int length)
{
  return null;
}
```

Test failed!

(nil/null->constant)

```
public static String wrap(String s, int length)
{
  return "";
}
```

Test passed!

```
@Test
public void OneShortWordDoesNotWrap()
{
  assertThat(wrap("word", 5), is("word"));
}
```

(unconditional->if)
+
(constant->scalar)

```
public static String wrap(String s, int length)
{
  if (s == null)
    return "";
  return s;
}
```

QUESTIONS?