



Behaviour-Driven Development

Utilisation de BDD par la pratique avec Cucumber



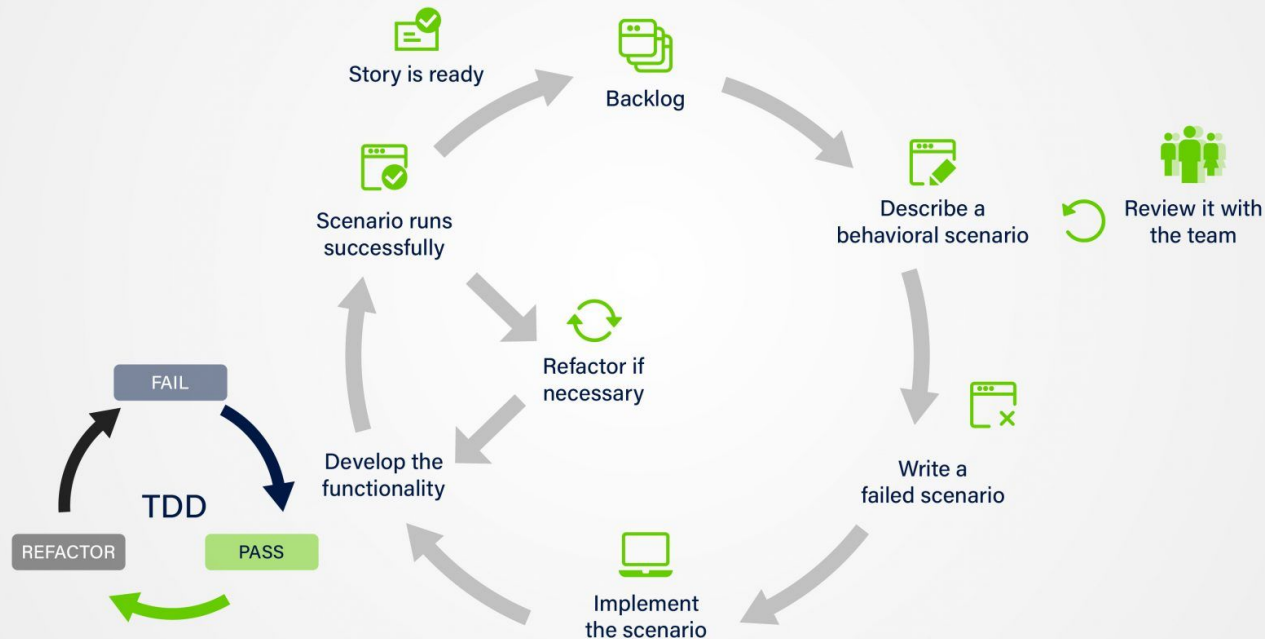
Le Behaviour-Driven Development ou développement piloté par le comportement

Approche inventée par Dan North et décrite pour la première fois dans un article de Better Software en 2006.

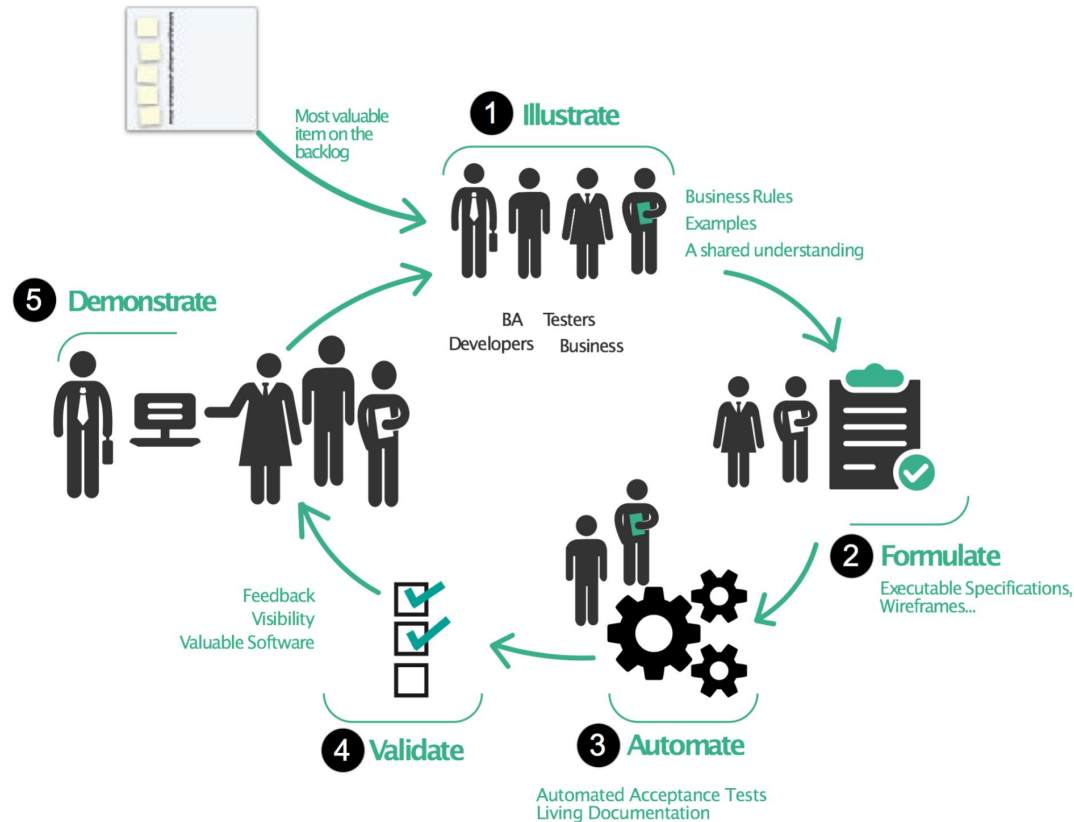
- On ne raisonne pas sur une unité du programme comme en Test-Driven Development (TDD) mais sur une approche plus globale et fonctionnelle
- On cherche l'expressivité du besoin par la description :
 - des contextes d'utilisation
 - des actions à effectuer sur le système
 - des critères d'acceptation du système
- On cherche la collaboration, la compréhension et la validation du système par différentes parties prenantes (Clients, PO - Product Owner, devs, équipes analyse qualité, etc...)

Le BDD et la pratique agile

The BDD Process



Le BDD et les parties prenantes

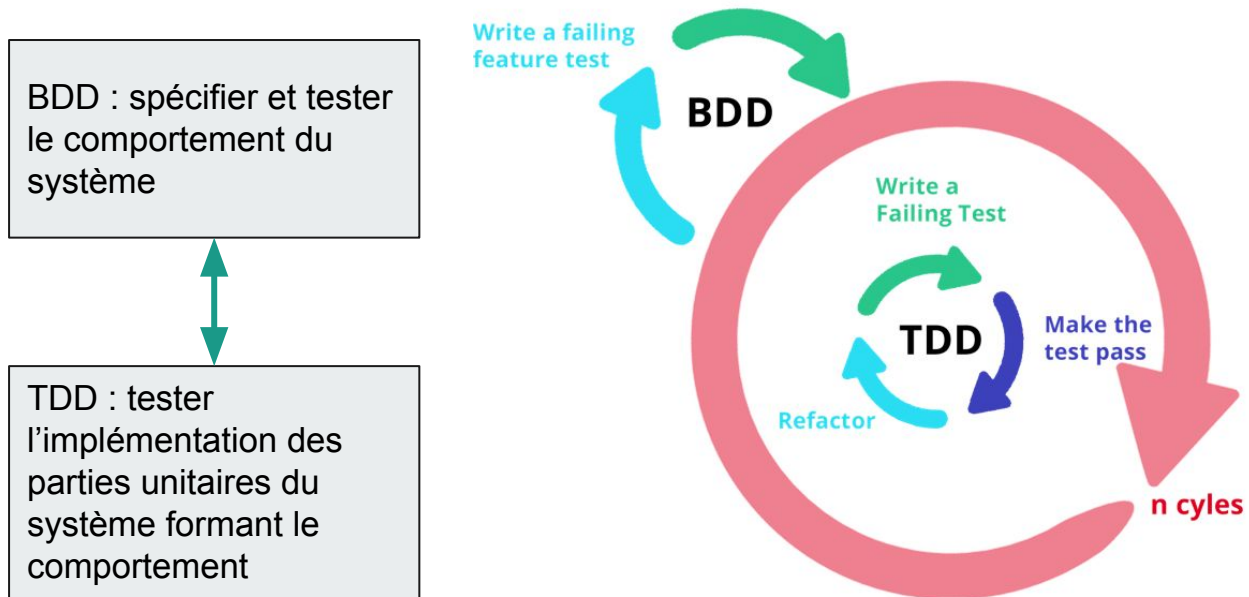




Différences BDD et TDD

	BDD	TDD
Parties prenantes	Développeurs, équipes QA , clients, PO, SM, etc	Développeurs, Scrum Master
Langage	Simplification (sous-ensemble) du langage naturel	Langage(s) de programmation lié à la /aux technologie(s) de test
Niveau d'implémentation	Haut-niveau (comportemental)	Bas-niveau (implémentation pure)
Etapes de developpement	Discussions des features/scénarios , implémentation, test et refactoring	Implémentation / test / refactoring
Documentation(s) utilisée(s)	Exigences du système, critères d'acceptation	Exigences d'implémentation

Mais BDD et TDD sont complémentaires



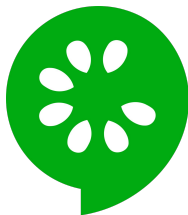
Source :
<https://www.digite.com/agile/behavior-driven-development-bdd/>



BDD : les outils



Quantum Automation



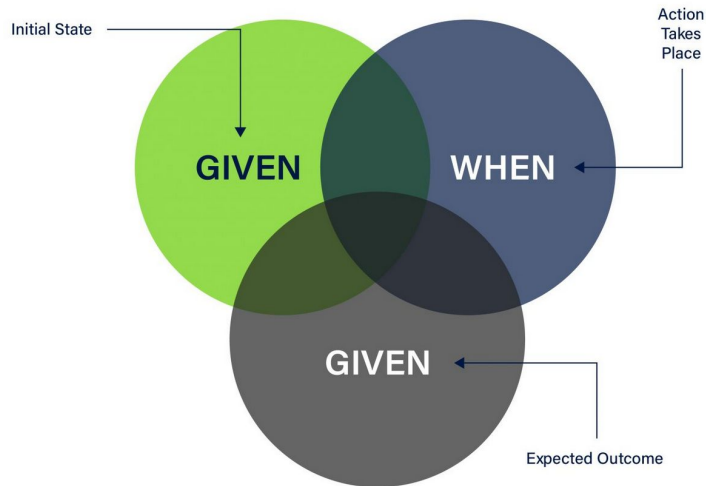
Formalisation du comportement en BDD par Gherkin avec Cucumber

Gherkin : syntaxe simplifiée du langage naturel permettant de décrire le comportement d'un système et son contexte

L'implémentation d'un test avec Cucumber est basée sur 3 éléments textuels permettant de décrire le comportement :

- **Given** : étant donné un contexte initial
- **When** : quand des événements ou actions surviennent
- **Then** : alors on attend un résultat spécifique

Cette formalisation s'écrit dans des fichiers .feature en Cucumber





Formalisation du comportement en BDD par Gherkin avec Cucumber

Gherkin : syntaxe simplifiée du langage naturel permettant de décrire le comportement d'un système et son contexte

L'implémentation d'un test avec Cucumber est basée sur 3 éléments textuels permettant de décrire le comportement :

- **Given** : étant donné un contexte initial
- **When** : quand des événements ou actions surviennent
- **Then** : alors on attend un résultat spécifique

Cette formalisation s'écrit dans des fichiers .feature en Cucumber

Le fichier JeuDuPendu.feature en français

```
#language: fr
```

```
Fonctionnalité: Proposer une lettre
```

```
Scénario: Connaître l'occurrence d'une lettre dans un mot
```

```
    Etant donné que le mot à trouver est cucumber  
    Quand le joueur propose la lettre u  
    Alors la lettre est présente 2 fois
```

```
Scénario: Connaître la position de toutes les occurrences dans un mot
```

```
    Etant donné que le mot à trouver est cucumber  
    Quand le joueur propose la lettre u  
    Alors le mot est -u-u----
```

Source : <https://www.digite.com/agile/behavior-driven-development-bdd/>



Formalisation du comportement en BDD par Gherkin avec Cucumber

Le fichier PlayHangman.feature

Feature: Suggesting a letter

Scenario: Knowing the occurrence of one letter in the word

Given the word to be guessed is cucumber
When the player suggests the letter u
Then the letter is found 2 times

Scenario: Knowing the position of all occurrences in the word

Given the word to be guessed is cucumber
When the player suggests the letter u
Then the word is -u-u----

Le fichier JeuDuPendu.feature en français

#language: fr

Fonctionnalité: Proposer une lettre

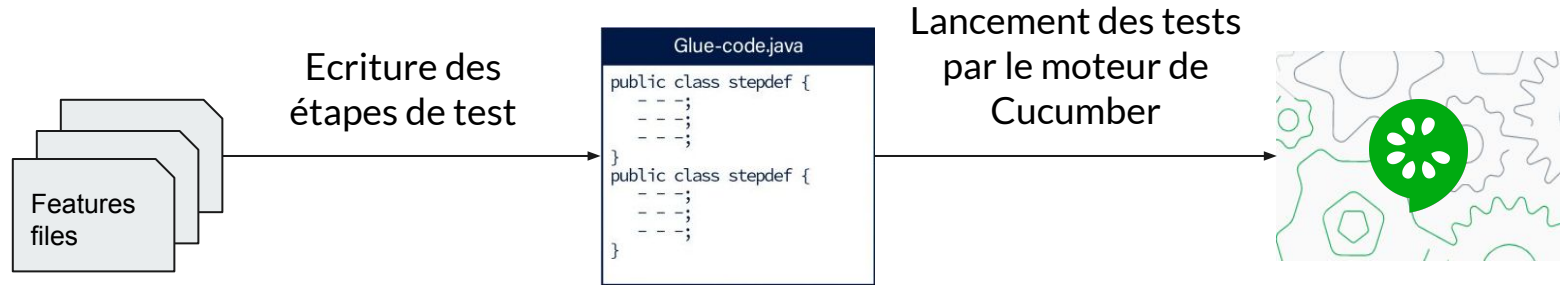
Scénario: Connaître l'occurrence d'une lettre dans un mot

Etant donné que le mot à trouver est cucumber
Quand le joueur propose la lettre u
Alors la lettre est présente 2 fois

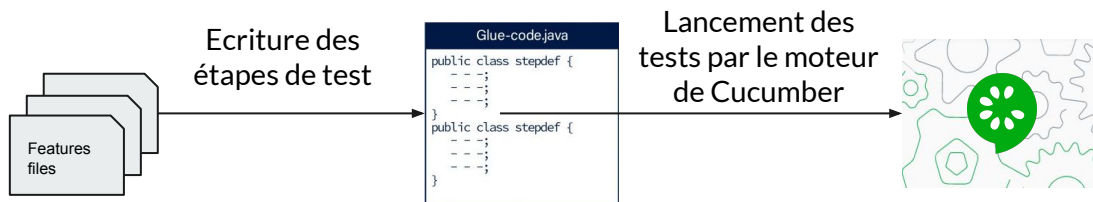
Scénario: Connaître la position de toutes les occurrences dans un mot

Etant donné que le mot à trouver est cucumber
Quand le joueur propose la lettre u
Alors le mot est -u-u----

Implémentation de BDD en Java avec Cucumber



Implémentation de BDD en Java avec Cucumber



```
Feature: Make a coffee with a complete coffee machine
A user want a coffee
Scenario: A user plug the coffee machine and make a coffee Arabica
  Given a coffee machine with 0.10 l of min capacity, 5.0 l of max capacity,
  And a "mug" with a capacity of 0.25
  When I plug the machine to electricity
  And I add 1 liter of water in the water tank
  And I add 0.5 liter of "ARABICA" in the bean tank
  And I made a coffee "ARABICA"
  Then the coffee machine return a coffee mug not empty
  And a coffee volume equals to 0.25
  And a coffee "mug" containing a coffee type "ARABICA"
```

```
2 usages  qperrez
@Given("a coffee machine with {double} l of min capacity, {double} l of max capacity, {double} l of max capacity")
public void givenACoffeeMachine(double minimalWaterCapacity, double maximalWaterCapacity, double maximalWaterCapacity) {
    coffeeMachine = new CoffeeMachine(minimalWaterCapacity, maximalWaterCapacity, minimalWaterCapacity);
}
```

```
2 usages  qperrez
@And("a {string} with a capacity of {double}")
public void aWithACapacityOf(String containerType, double containerCapacity) {
    if ("mug".equals(containerType)) {
        mug = new Mug(containerCapacity);
    }
    if ("cup".equals(containerType)) {
        cup = new Cup(containerCapacity);
    }
}
```

```
2 usages  qperrez
@When("I plug the machine to electricity")
public void iPlugTheMachineToElectricity() { coffeeMachine.plugToElectricalPlug(); }
```

```
2 usages  qperrez
@And("I add {double} liter of water in the water tank")
public void iAddLitersOfWater(double waterVolume) { coffeeMachine.addWaterInTank(waterVolume); }
```

```
@RunWith(Cucumber.class)
@CucumberOptions(
    features = {"classpath:functional/features/"},
    glue = "fr.imt.coffee.machine.cucumber.steps"
)
//Permet d'ignorer les tests fonctionnels de Cucumber
//Ne lance pas la class CoffeeMachineFunctionalTest
public class CoffeeMachineCucumberFunctionalTest {
}
```

Implémentation de BDD en Java avec Cucumber

```
Feature: Make a coffee with a complete coffee machine
  A user want a coffee
  Scenario: A user plug the coffee machine and make a coffee Arabica
    Given a coffee machine with 0.10 l of min capacity, 3.0 l of max capacity,
    And a "mug" with a capacity of 0.25
    When I plug the machine to electricity
    And I add 1 liter of water in the water tank
    And I add 0.5 liter of "ARABICA" in the bean tank
    And I made a coffee "ARABICA"
    Then the coffee machine return a coffee mug not empty
    And a coffee volume equals to 0.25
    And a coffee "mug" containing a coffee type "ARABICA"
```

```
2 usages  qperez
@Given("a coffee machine with {double} l of min capacity, {double} l of max capacity, {double} l of max capacity")
public void givenACoffeeMachine(double minimalWaterCapacity, double maximalWaterCapacity, double maximalWaterCapacity) {
    coffeeMachine = new CoffeeMachine(minimalWaterCapacity, maximalWaterCapacity, minimalWaterCapacity);
}
```

```
2 usages  qperez
@And("a {string} with a capacity of {double}")
public void aWithACapacityOf(String containerType, double containerCapacity) {
    if ("mug".equals(containerType))
        mug = new Mug(containerCapacity);
    if ("cup".equals(containerType))
        cup = new Cup(containerCapacity);
}
```

```
2 usages  qperez
@When("I plug the machine to electricity")
public void iPlugTheMachineToElectricity() { coffeeMachine.plugToElectricalPlug(); }
```

```
2 usages  qperez
@And("I add {double} liter of water in the water tank")
public void iAddLitersOfWater(double waterVolume) { coffeeMachine.addWaterInTank(waterVolume); }
```

Et voilà !

