

Cucumber et BDD

(BEHAVIOUR DRIVEN DEVELOPMENT)

Autom / DTA

Introduction au Behaviour Driven Developpement (BDD)

PARTIE I : Introduction au Behaviour Driven Developpement (BDD)

- Présentation comparée des différentes approches de développement
- Les principes généraux de l'Agile
- Testing first - les méthodes de développement Agile
- Le BDD : acteurs et workflow
- Les outils du BDD - langage Gherkin et framework d'implémentation



Introduction au Behaviour Driven Development (BDD)

Présentation comparée des différentes approches de développement

Les méthodes et pratiques de développement se calquent sur la gestion de projet dans lesquelles elles s'inscrivent.

On distingue deux grands types d'approches :

APPROCHE
TRADITIONNELLE



prédictive et séquentielle

APPROCHE AGILE



itérative et adaptative

L'approche traditionnelle	L'approche agile
Elle suppose de <i>spécifier et planifier dans les détails</i> l'intégralité d'un produit avant de le développer.	Elle consiste à <i>définir un objectif</i> , puis apprendre en faisant, en procédant par étape
Les activités (spécification, conception, implémentation, vérification) sont séquentielles	Spécification, conception et vérification s'enchaînent au cours de phases itératives de courte durée
Les testeurs interviennent à la fin du cycle de développement	Les testeurs sont parties prenantes des développements
Méthodes de gestion de projet : Cycle en V, Cascade...	Méthodes de gestion de projet : Scurm, Kanban, XP...

Introduction au Behaviour Driven Developpement (BDD)

Les principes généraux de l'Agile

Les pratiques de développement itératives et incrémentales ne sont pas récentes (les premières apparaissent dans les années 1970), mais c'est le Manifeste Agile, rédigé en 2001, qui a consacré le terme « Agile » pour qualifier ces approches.

Ce manifeste comporte uniquement une déclaration de 4 valeurs et 12 principes. On parle de méthodes agiles pour les méthodes fondées sur ces principes.

« *Ces expériences nous ont amenés à valoriser :*



« *Nous reconnaissons la valeur des seconds éléments, mais privilégions les premiers.* »

Les trois méthodes agiles les plus utilisées sont : **Kanban**, **Scrum**, et **XP Extreme programming**.

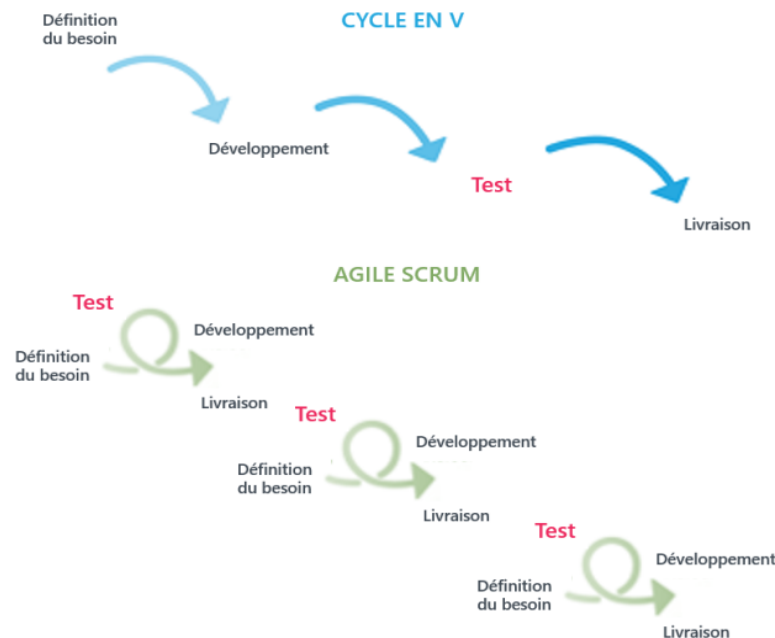
Introduction au Behaviour Driven Development (BDD)

Les principes généraux de l'Agile

Dans les approches de développement traditionnelles, le produit est présenté au client dans un état très avancé (voire terminé). → risque d'**effet tunnel**.

Au contraire l'approche agile consiste à adopter un processus itératif développement : chaque incrément inclut une liste de fonctionnalités à développer, des travaux de spécification, de développement et de test.

A la fin de chaque itération on obtient donc un produit viable, utilisable → "**Minimum viable product**".



BENEFICES

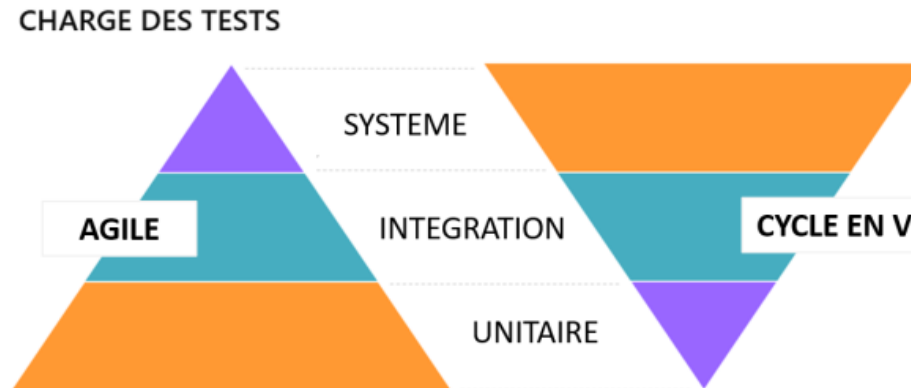
- Lever les risques tôt si le produit n'évolue pas dans le bon sens,
- Eviter les incompréhensions sur les besoins (qui auraient été décelés plus tard dans un projet traditionnel, et auraient alors été plus coûteux à rectifier),
- Recevoir des feedbacks client et utilisateurs très tôt et fréquemment

Introduction au Behaviour Driven Development (BDD)

Testing first – quelques principes du développement Agile

En contexte agile, l'activité de test est présente à tous les niveaux : **unitaire, intégration, système, acceptation.**

Mais, la logique adoptée diffère des cycles de développement traditionnel. C'est le **Testing first**



« plus une anomalie est découverte tard, plus sa correction coûte cher »

Trois méthodes pour tester le code au plus tôt (XP-Programming) :

- Les test driven developpement (TDD)
- Le pair programming
- L'intégration continue

Introduction au Behaviour Driven Development (BDD)

TESTER LE CODE : Le TDD (test driven developpement)

Le principe du TDD va à **contre-courant de la logique de test traditionnelle**.

L'idée n'est pas de tester un code déjà implémenté, mais de partir de l'écriture du test (méthode de test unitaire), pour ensuite écrire le code qui permettra au test de passer en succès



Introduction au Behaviour Driven Development (BDD)

TESTER LE CODE : Paire Programming

La programmation se fait à deux

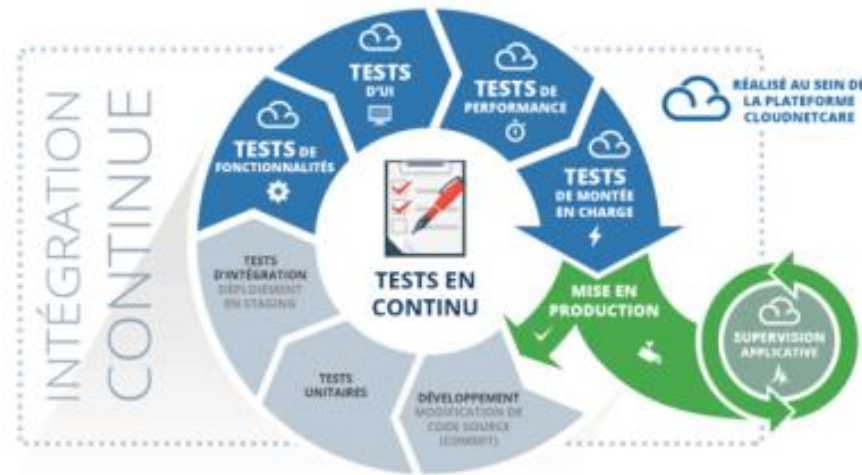
- L'un est le **driver**. C'est lui qui tient le clavier.
- Le second, appelé **partner** (ou copilote) est là pour l'aider en suggérant de nouvelles possibilités ou en décelant d'éventuels problèmes.
- Changent fréquemment de partenaire
 - améliorer la connaissance collective de l'application
 - améliorer la communication au sein de l'équipe



Introduction au Behaviour Driven Developpement (BDD)

TESTER LE CODE : Intégration continue

Chaque modification du code source donne lieu à une vérification automatisée de sa qualité.



L'intégration continue dans sa forme simple: Consiste à surveiller les modifications dans le code : dès qu'un changement est détecté, le serveur d'exécution compile et teste le code (unitaire)

L'intégration continue dans sa forme complexe: Le serveur d'exécution opère régulièrement des analyses qualimétriques. Il compile, teste le code et déploie l'application. Passe les tests système (TNR, performance, stress, sécurité...) et crée un reporting public (client, développeurs, MOA & AMOA)

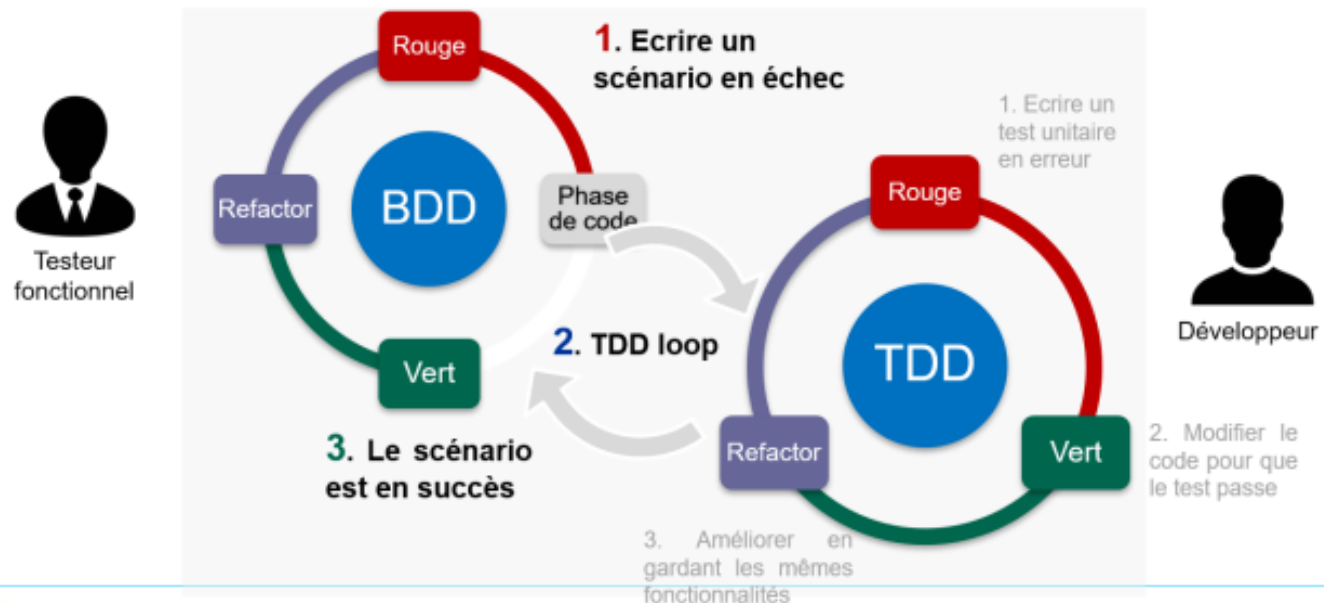
Introduction au Behaviour Driven Developpement (BDD)

TESTER LES FONCTIONNALITÉS : Behaviour Driven Developement (BDD)

Comme le TDD, le Behaviour Driven Developement (BDD) formalise la logique d'écriture des tests avant l'implémentation du code, mais au niveau système :

- Le BDD se concentre sur l'**aspect comportemental** attendu du système et non sur les fonctions du code.
- Le BDD donne une meilleure compréhension de ce que le système devrait faire du point de vue du client, puisqu'il utilise **un langage naturel** plutôt qu'informatique pour les *scenarii*.
- Le BDD permet au développeur et au client de **travailler en commun** sur l'analyse des exigences

Le besoin fonctionnel guide le développement de l'application



Introduction au Behaviour Driven Developpement (BDD)

Le BDD : les acteurs

Dans un projet Agile, le BDD va formaliser la communication entre tous les membres d'une équipe

Les rôles en Agile (ex : une équipe Scrum) :

Le product owner : Il porte la vision du produit à réaliser. Il doit faire en sorte qu'un maximum de valeur soit produite par l'équipe en faisant en sorte que les fonctionnalités importantes soient priorisées comme telles. Il est responsable de la description des user stories et doit s'assurer de la bonne compréhension de l'équipe...

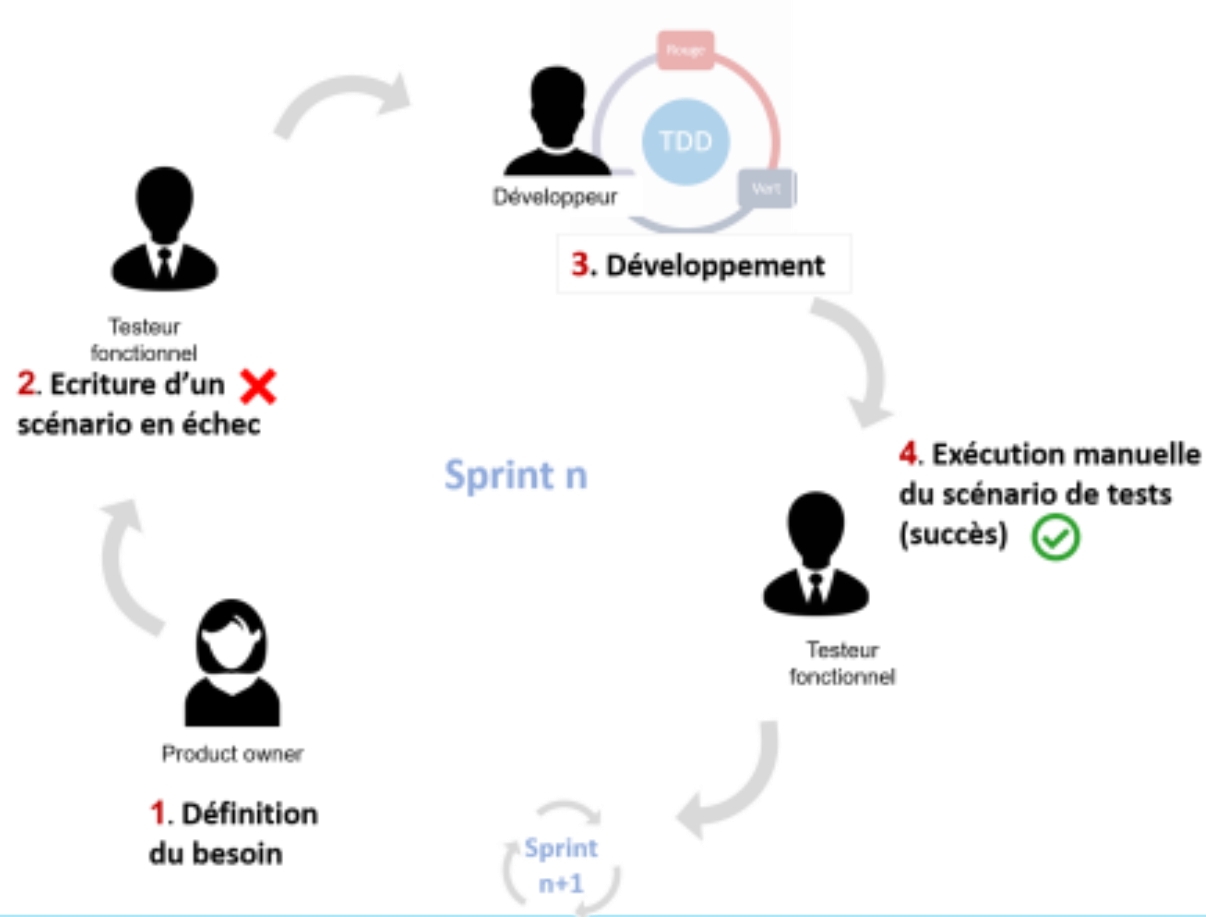
Le scrum master : Il est le facilitateur, celui qui s'assure que Scrum est compris et accepté par toute l'équipe (il insuffle et facilite, il est à l'écoute de l'équipe et prend toutes les dispositions possibles pour faciliter la progression de l'équipe).

L'équipe : Elle est composée des personnes qui travaillent pour livrer un incrément « Fini » et potentiellement livrable en production. Elle s'auto-organise pour créer des incréments de produit fonctionnels et de qualité. Elle se compose de testeurs (fonctionnels et/ou techniques) et de développeurs.

Introduction au Behaviour Driven Development (BDD)

Le BDD : le workflow

Dans la logique Agile BDD, le scénario est spécifié avant le développement. A la fin du sprint, si tout s'est passé comme convenu, les *scenarii* sont en succès



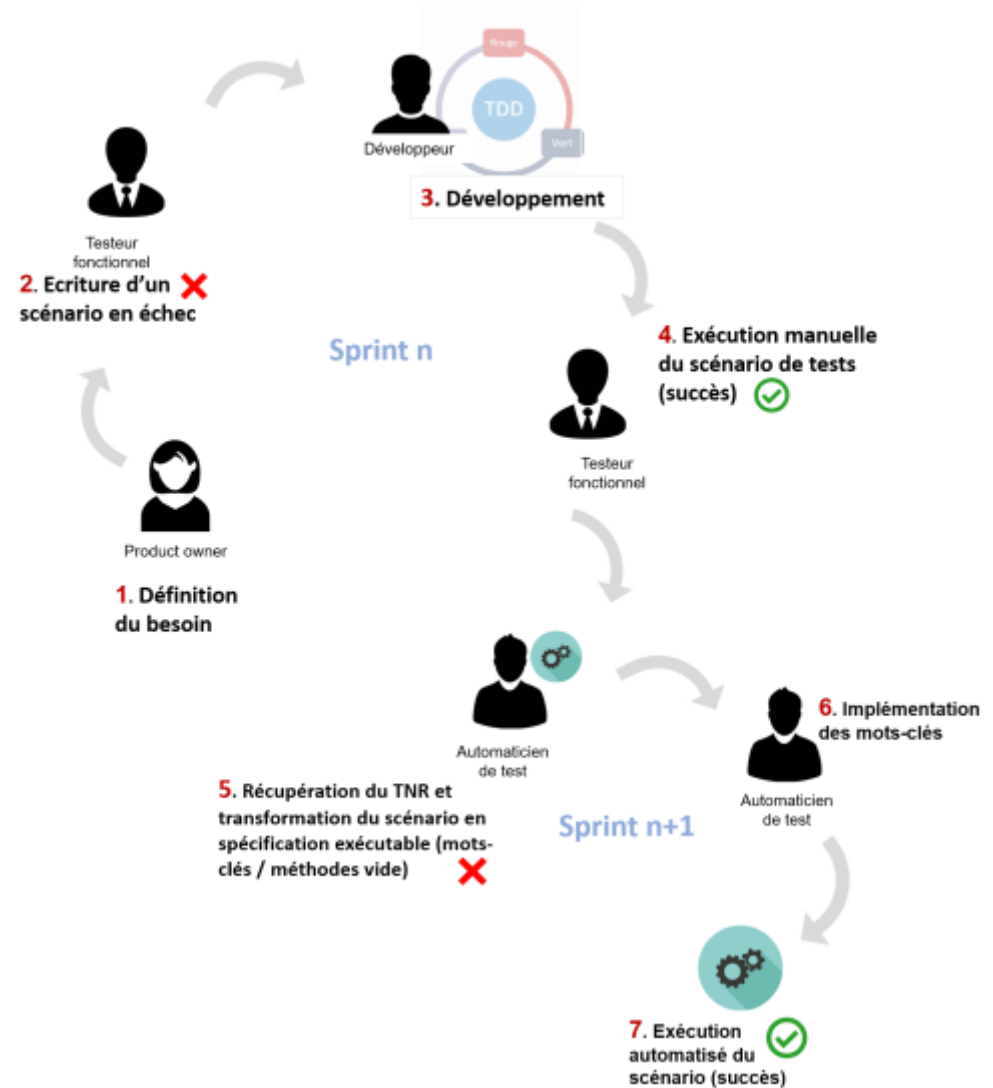
Introduction au Behaviour Driven Developpement (BDD)

Le BDD : le workflow

Automatiser les TNR en BDD

Une fois le scénario en succès, l'équipe aura pour tâche d'automatiser son exécution.

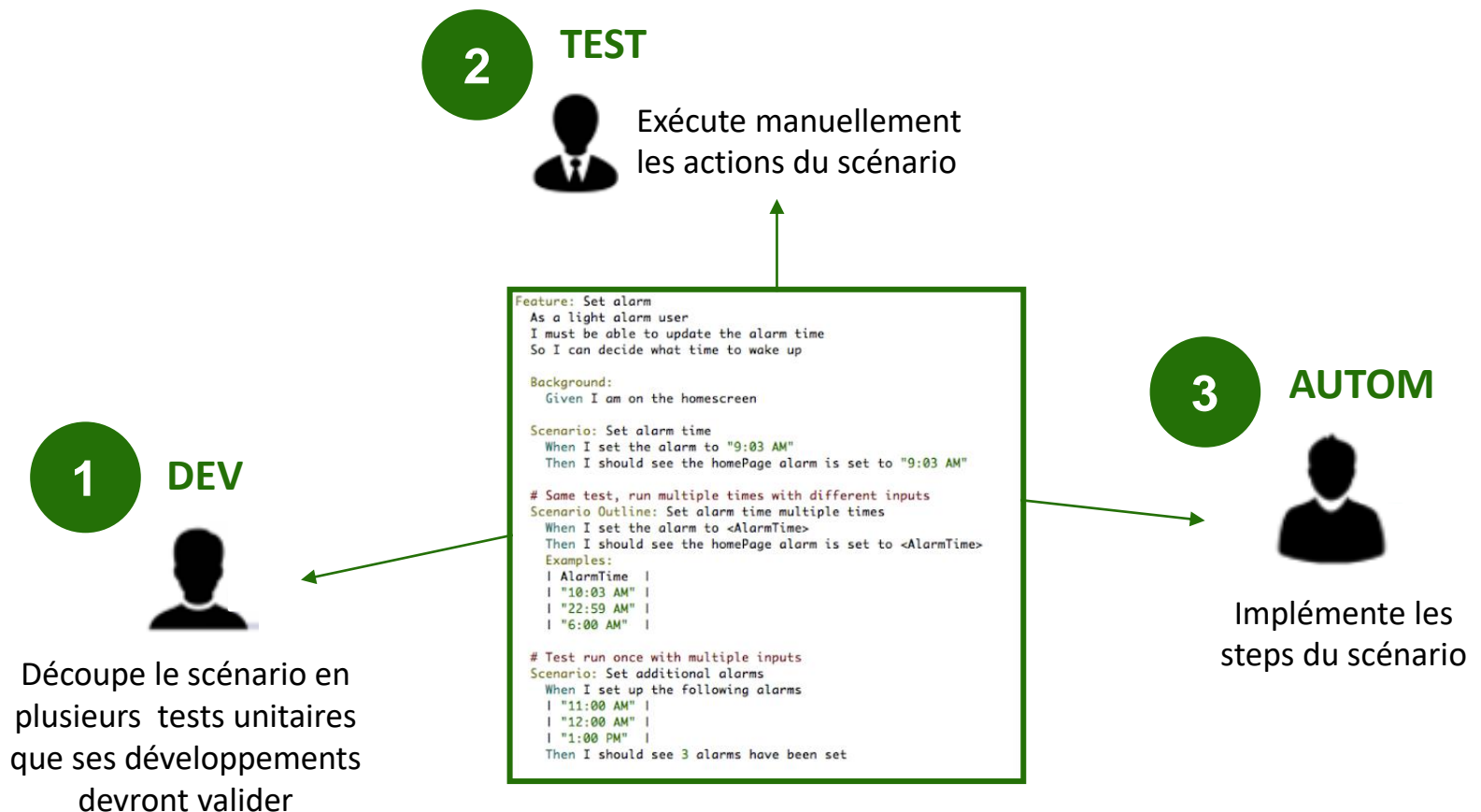
Le scénario, s'il est bien spécifié, deviendra le script de test automatisé, à condition qu'un automaticien implémente les mots-clés.



Qu'est-ce que le BDD ?

le scénario Gherkin

Le scénario Gherkin est au centre des activités de développement, de vérification et d'automatisation



Introduction au Behaviour Driven Developpement (BDD)

Les outils du BDD - langage Gherkin

Qu'est-ce que Gherkin ?

- Langage dédié à l'écriture de scénario BDD en vue d'une automatisation (DSL).
- ≠ Des tests fonctionnels en cycle en V (action / résultat attendu),
- les scenarii de test BDD se calquent sur le formalisme des User Stories.
- Les fonctionnalités sont décrites du point de vue de l'utilisateur plutôt que du point du vue logiciel.

Anatomie d'une User Story

La description

Décrit un besoin ou une attente exprimée du point de vue d'un utilisateur

AS En tant que 'un utilisateur'

I WANT TO Je veux 'une action'

IN ORDER TO Afin de 'But de la story'

Les critères d'acceptation

Ensemble de conditions que la story doit satisfaire pour être considérée comme complète et terminée.

GIVEN : Qui décrit l'état initial de l'application
(Prérequis du cas de test)

WHEN : Qui décrit un événement ou un moment précis
(Action du cas de test)

THEN : Qui précise la réaction du SI à l'événement du
WHEN (Résultat attendu du cas de test)

Introduction au Behaviour Driven Development (BDD)

Les outils du BDD - langage Gherkin

Le langage Gherkin se calque sur la structure des critères d'acceptation des *Users Story* en utilisant les mots-clés **GIVEN-WHEN-THEN.**

Feature: Se connecter à l'espace personnel

Scenario : Se connecter avec succès

Given Je suis un client de la banque

When je me connecte au site avec les identifiants « **user1** » et « **password1** »

Then j'accède à mon espace personnel

Parce qu'ils s'écrivent avec des mots-clefs spécifiques, les scripts Gherkin respectent un formalisme qui facilite leur automatisation.

Introduction au Behaviour Driven Developpement (BDD)


Les outils du BDD – le framework Cucumber

Des spécifications exécutables

Les frameworks d'implémentation BDD servent à automatiser l'exécution d'un scénario de test. Ils agissent comme des parseurs qui interprètent les mots-clés Gherkin (**GIVEN, WHEN, THEN**) comme des méthodes (fonctions) que l'automaticien de test doit implémenter.

```
Feature: Vérifier la connexion d'un client

Scenario: Client enregistré
Given Je suis un client de la banque
And Je possède un compte en ligne
When je me connecte au site avec un login et un mot-de-passe valides
Then j'accède à mon espace personnel
```



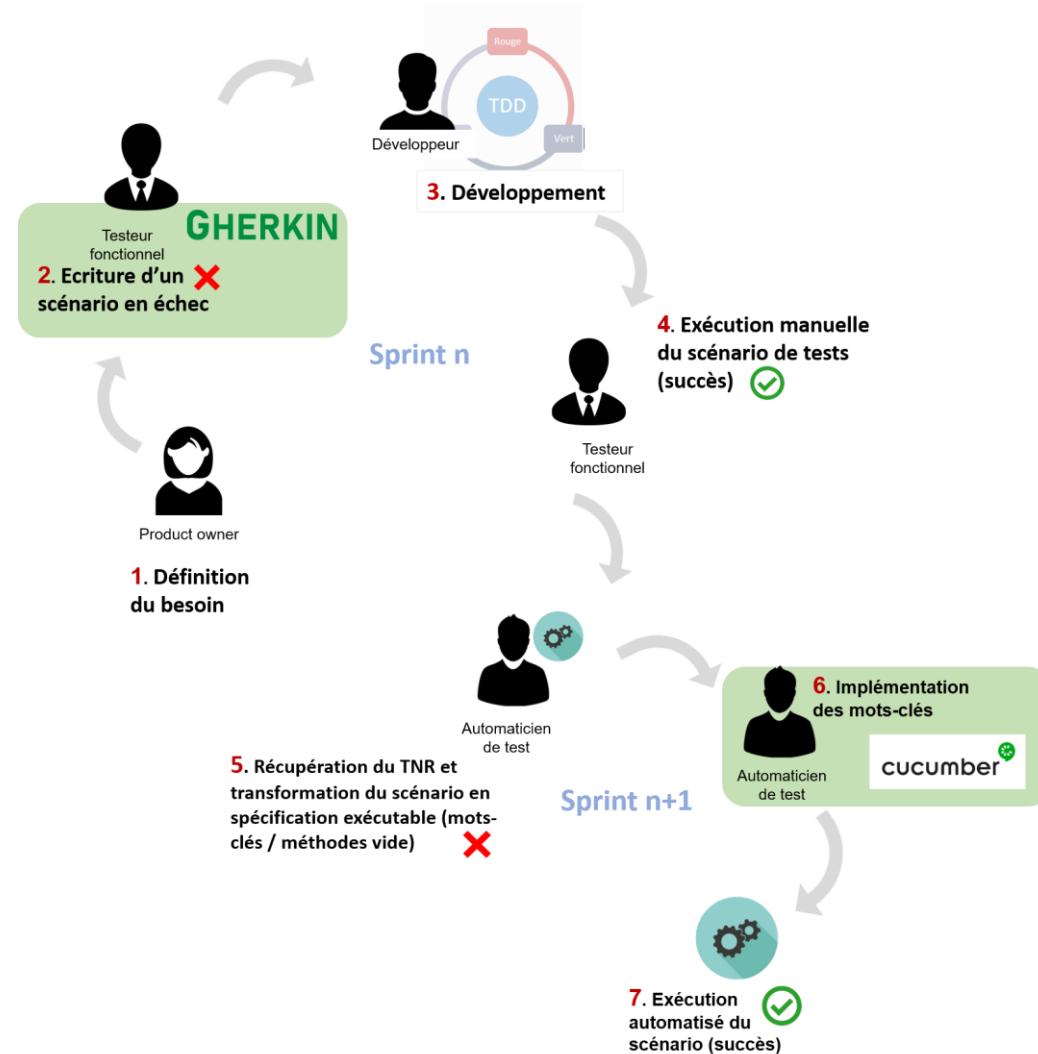
```
@When("je me connecte au site avec un login et un mot-de-passe valides")
public void je_me_connecte_au_site_avec_un_login_et_un_mot_de_passe_valides() {
    // Ecrire le code pour implémenter l'action
    throw new cucumber.api.PendingException();
}
```

Gherkin étant agnostique, l'implémentation des méthodes peut être réalisée dans n'importe quel langage de programmation et à partir de différents framework :

- **Cucumber** (Java, Ruby, JavaScript) et ses extensions : **Behave** (Python), **CukeSalad** (orienté ATDD : task, goal, action)
- **Jbehave** (Java)
- **Turnip** (Ruby)
- **Spinach**

Introduction au Behaviour Driven Development (BDD)

Les outils du BDD – le framework Cucumber



PARTIE 2 : Implémenter les scripts Gherkin

- Gherkin syntaxe basique
- Automatiser : l'implémentations des mots-clés
- Automatiser : Création d'un projet Maven Cucumber
- Automatiser : La gestion des données



Utilisation de Gherkin / Cucumber

Gherkin syntaxe basique

1°. Un cas de test Gherkin teste UNE SEULE fonctionnalité

Un script Gherkin correspond à une fonctionnalité. C'est d'ailleurs par ce mot-clé que commence le script :

En français : Fonctionnalité

En anglais : Feature

2°. Une fonctionnalité se vérifie par UN ou PLUSIEURS scénarios

Un script Gherkin est constitué de 1 à n scénarios. Ces scénarios sont autant de tests visant à valider la fonctionnalité.

En français : Scénario

En anglais : Scenario

3°. Un scénario DOIT contenir les trois mots clés GIVEN, WHEN, THEN

Given, When, Then constituent les trois étapes obligatoires d'un scénario Gherkin

Ces mots-clés peuvent se répéter selon si l'étape se compose d'une ou plusieurs conditions, actions ou vérifications

Utilisation de Gherkin / Cucumber

Gherkin syntaxe basique – Anglais / Français

Anglais	Français
Given	Soit Etant donné que Etant donné qu' Etant données Etant donnés Etant donnée Etant donné
When	Quand Lorsque Lorsqu'
Then	Alors

EXEMPLE

`# language: fr`

`Fonctionnalité: commander une place de concert`

`Scénario: Je commande une place avec un solde suffisant`

`Etant donné que je suis un utilisateur connecté`

`Etant donné que mon solde est de 30 euros`

`Quand je mets dans mon panier d'achat un billet de concert au prix de 25 euros`

`Alors un message confirme la transaction`

`Alors je peux consulter mon billet dans la rubrique achat`

Utilisation de Gherkin / Cucumber

Exercice

En utilisant les mots clés Gherkin anglais, complétez ce script.

: Authentification

Scenario : Vérifie la bonne connexion d'un utilisateur enregistré

je me trouve sur la page d'authentification du site

je dispose d'un compte utilisateur « ginette »

je saisis mon identifiant « ginette » dans le champ « Login »

When je saisis mon mot de passe « secret » dans le champ « Password »

je clique sur le bouton « Connexion » du formulaire

je suis authentifié sur le site

je suis redirigé vers la page « Mon compte »

Utilisation de Gherkin / Cucumber

Gherkin syntaxe basique – Les AND et BUT

And et But sont deux mots-clés permettant une meilleure lisibilité des scripts Gherkin lorsque ceux-ci se composent de condition ou vérifications multiples.

- **AND** est utilisé pour des conditions, actions ou vérifications additionnelles **POSITIVES** (en français : Et ; Et que ; Et qu')
- **BUT** est utilisé pour des conditions, actions ou vérifications additionnelles **NÉGATIVES** (en français : Mais ; Mais que ; Mais qu')

```
Scenario: without and & but
  Given L'utilisateur est sur la page de connexion
  When l'utilisateur entre "ShankarGarg" dans le champ username
  When l'utilisateur entre "123456" dans le champ password
  When l'utilisateur clic sur le bouton Submit
  Then La page d'accueil s'affiche
  Then Le bouton Submit n'est pas présent
```

```
Scenario: with and & but
  Given L'utilisateur est sur la page de connexion
  When l'utilisateur entre "ShankarGarg" dans le champ username
  And l'utilisateur entre "123456" dans le champ password
  And l'utilisateur clic sur le bouton Submit
  Then La page d'accueil s'affiche
  But Le bouton Submit n'est pas présent
```

Utilisation de Gherkin / Cucumber

Gherkin syntaxe basique – Le Contexte (Background) 1/3

Le mot-clé Contexte (Background) permet de préciser les prérequis du ou des scénarios composant votre script.

- Puisqu'il s'agit d'un prérequis, il sera souvent constitué d'un simple Given.

```
# language: en
Feature: Commander une place de concert

  Background: L'utilisateur est connecté
    Given Je suis un utilisateur connecté au site de e-commerce

  Scenario: L'utilisateur commande une place avec un solde insuffisant
    Given J'ai un solde de 20 euros
    When J'ajoute à mon panier un billet de concert au prix de 25 euros
    Then Un message d'alerte s'affiche indiquant que mon solde est insuffisant
```

| prérequis

| test

Utilisation de Gherkin / Cucumber

Gherkin syntaxe basique – Le Contexte (Background) 2/3

- Il se peut néanmoins que le Background prenne une forme plus complexe si le prérequis implique lui-même un premier test.

```
# language: en
Feature: Commander une place de concert

  Background: L'utilisateur est connecté
  Given Je suis sur la page de connexion du site e-commerce
  When Je sou mets mon login et mon mot-de-passe
  Then Je suis connecté

  Scenario: L'utilisateur commande une place avec un solde insuffisant
  Given J'ai un solde de 20 euros
  When J'ajoute à mon panier un billet de concert au prix de 25 euros
  Then Un message d'alerte s'affiche indiquant que mon solde est insuffisant
```

prérequis

test

Utilisation de Gherkin / Cucumber

Gherkin syntaxe basique – Le Contexte (Background) 3/3

- Dans le cas d'un script composé de plusieurs scénarios, le Background s'appliquera à tous les scénarios du script.
- → Lorsque les mots-clés seront implémentés, le test automatisé jouera systématiquement le Background avant chacun des scénarios

```
# language: en
Feature: Commander une place de concert

  Background: L'utilisateur est connecté
    Given Je suis un utilisateur connecté au site de e-commerce

  Scenario: L'utilisateur commande une place avec un solde insuffisant
    Given J'ai un solde de 20 euros
    When J'ajoute à mon panier un billet de concert au prix de 25 euros
    Then Un message d'alerte s'affiche indiquant que mon solde est insuffisant

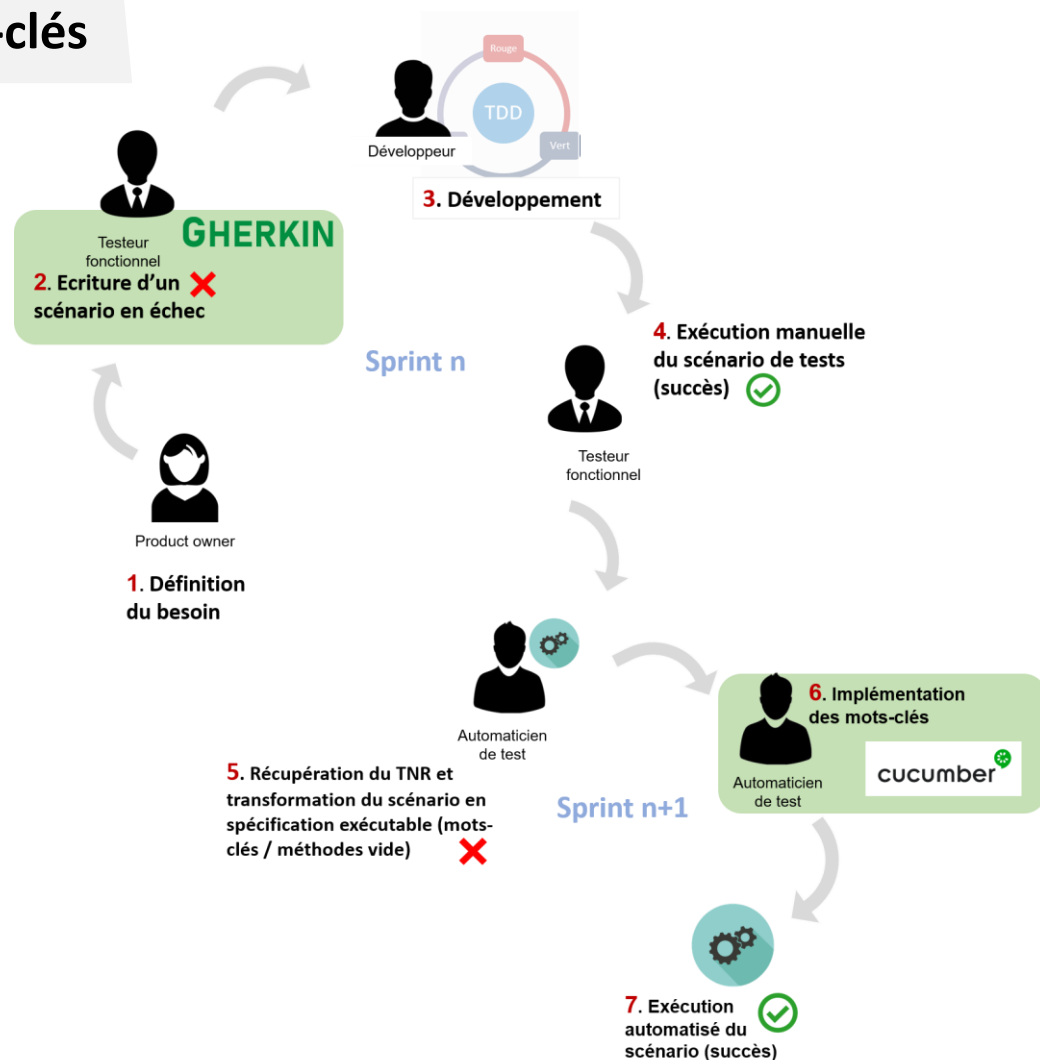
  Scenario: L'utilisateur commande une place avec un solde suffisant
    Given J'ai un solde de 30 euros
    When J'ajoute à mon panier un billet de concert au prix de 5 euros
    Then Je parviens à acheter la place de concert
```

prérequis

test

Utilisation de Gherkin / Cucumber

L'implémentation des mots-clés



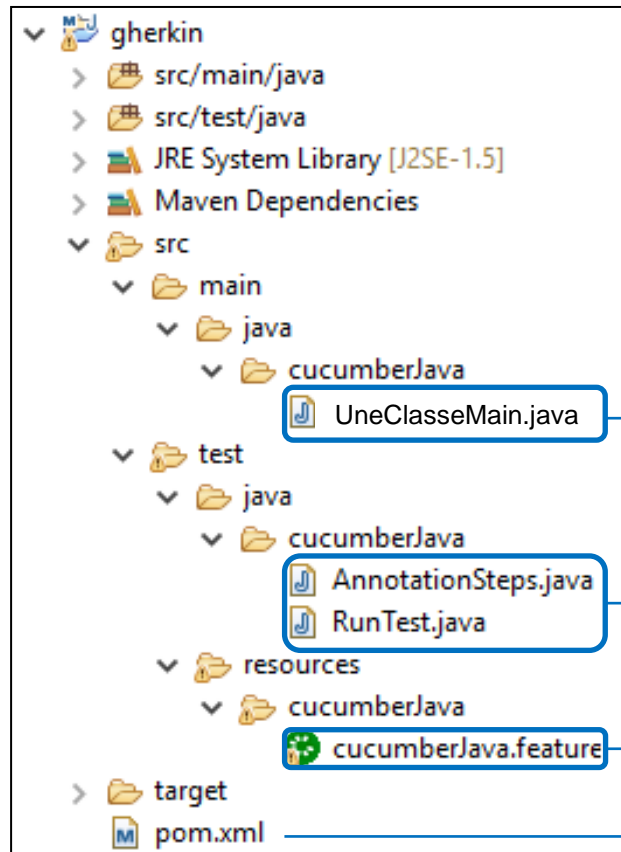
L'implémentation est le travail de l'automaticien.

Il utilise un framework (ex : Cucumber) capable :

- d'interpréter le langage Gherkin
- de faire le lien entre un script Gherkin et les méthodes implémentées
- d'exécuter un script Gherkin (les actions et vérifications automatisées)
- de générer un rapport d'exécution

Création d'un projet Maven

Télécharger le plugin Cucumber



Classes Main

Classes de tests

Scenario Gherkin

Mise à jour du pom nécessaire

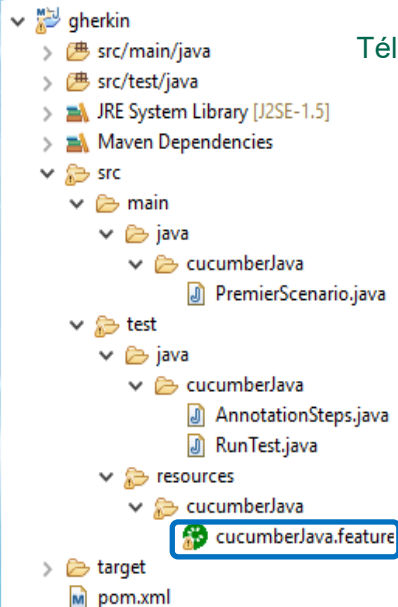
1° étape : Rajouter ces dépendances au pom.xml

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.6.0</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>5.6.0</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>5.6.0</version>
  </dependency>
</dependencies>
```

2° étape : Ecrire son scénario (feature) dans un fichier *.feature*

Le fichier *.feature* est à enregistrer sous ***src/test/resources/{package}***

Télécharger le plugin Cucumber Eclipse disponible sur marketplace pour bénéficier de l'aide syntaxique



Feature: Floorball player's recrutement

Scenario: I want to know more about the nearest floorball club

Given I have opened a browser

And I search for floorball France

When I click on the French floorball federation website

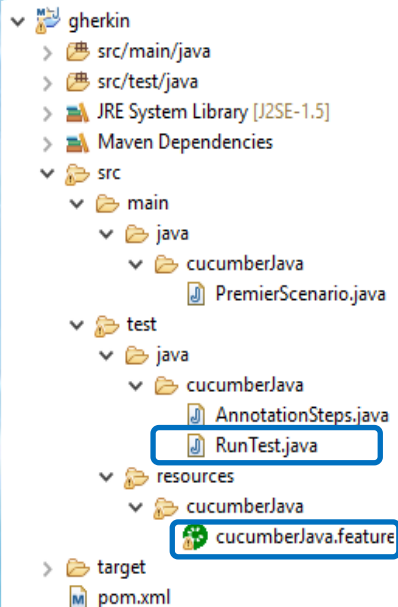
Then the menu où pratiquer is clickable



Majuscule aux mots-clés, pas d'espaces entre « **Feature** » / « **Scenario** » et « : »

3 ° Etape : Créer la classe de lancement des tests

Il s'agit d'un fichier .java à enregistrer sous *src/test/java/<package>*



```
package cucumberJava;
```

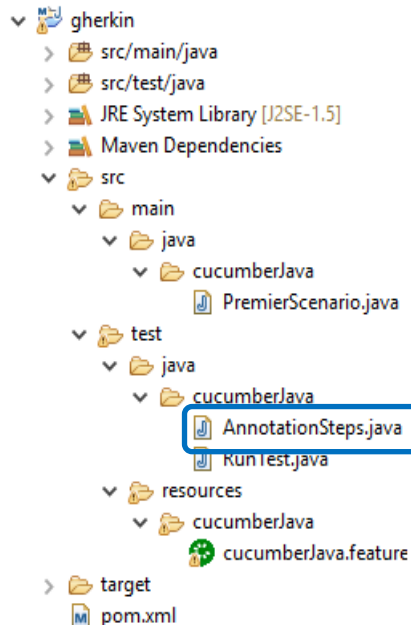
```
import org.junit.runner.RunWith;  
import io.cucumber.junit.*;
```

```
@RunWith(Cucumber.class)  
public class RunTest {  
}
```

A laisser vide !

4° Etape : Implémenter le code (« steps »)

- Selon le principe du BDD, on lance le test. L'exécution est en erreur, puisque le code n'est pas encore écrit...
- Le framework Cucumber propose alors un template de code en console.
➔ À récupérer et à placer dans une nouvelle classe : de préférence “*Steps.java”):



```
@Given("I have opened a browser")
public void i_have_opened_a_browser() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Given("I search for floorball France")
public void i_search_for_floorball_France() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("I click on the french floorball federation website")
public void i_click_on_the_french_floorball_federation_website() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("the menu ou pratiquer is clickable")
public void the_menu_is_clickable() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```


5° Etape : Modifier les méthodes Steps pour que le test soit en succès

```
public class AnnotationSteps {

    WebDriver driver;

    @Given("I have opened a browser")
    public void i_have_opened_a_browser() throws Throwable {
        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);
        driver.get("https://www.google.fr");
        driver.findElement(By.id("zV9nZe")).click();
    }

    @Given("I search for floorball France")
    public void i_search_for_floorball_France() throws Throwable {
        driver.findElement(By.xpath("//input[@class='gLfyf gsfi']")).sendKeys("France Floorball");
        driver.findElement(By.xpath("//div[@class='FPdoLc tfBOBf']/input[@name='btnK']")).submit();
    }

    @When("I click on the french floorball federation website")
    public void i_click_on_the_french_floorball_federation_website() throws Throwable {
        driver.findElement(By.xpath("//div/a[contains(@href,'www.floorball.fr')]/h3")).click();
    }

    @Then("the menu ou pratiquer is clickable")
    public void the_menu_is_clickable() throws Throwable {
        driver.findElement(By.xpath("//div[@id='colgauche']/div[2]/p[7]/a/img")).click();
    }
}
```

6° Etape : Lancer les tests

Succès



1 Scenarios ([32m1 passed[0m)
4 Steps ([32m4 passed[0m)
0m13,099s



Utilisation de Gherkin / Cucumber

La gestion des données

Il est important de distinguer au sein d'un script Gherkin ce qui est de l'ordre de la spécification invariable et ce qui est de l'ordre de la donnée de test.

Ces données deviendront des paramètres variables de nos méthodes

Les données de test Gherkin peuvent se présenter sous trois formes :

- Des chaînes de caractère ou caractères numériques
- Des « plan de scénario »
- Des tables de données



Le choix de l'écriture de ces données influe sur l'implémentation des méthodes d'automatisation

Utilisation de Gherkin / Cucumber

La gestion des données – chaîne de caractère et caractères numériques

Par défaut, un éditeur Gherkin signale toutes valeurs numériques ou chaîne de caractère entre guillemet à la manière d'une variable (colorimétrie)

→ Cucumber les considérera comme des variables de type entier ou String

```
Scenario: L'utilisateur commande un café
Given J'ai un solde de 1,50 euros
When Je sélectionne "un café sans sucre"
Then La machine prépare "un café sans sucre"
And Mon solde est désormais de 1,10 euros
```

— Donnée numérique

— Chaîne de caractère

```
...

@Given("J'ai un solde de {float} euros")
public void setSolde(float solde) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("Je sélectionne {string}")
public void getProduct(String product) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

...
```

Exercices Cucumber

Exercice 1.1

Formalisez un test écrit en Gherkin à partir du cas de test suivant



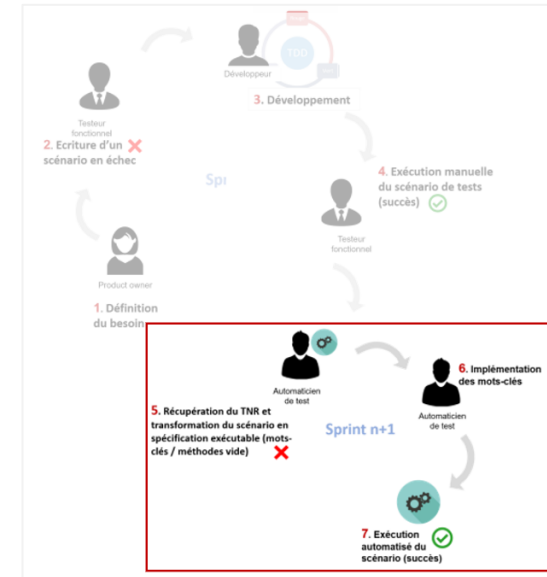
Actions	Résultat attendu
Accéder à l'application Jpetstore	Affichage de la page d'accueil de l'application
Se connecter en tant que j2ee/j2ee	L'utilisateur « ABC » est bien connecté (apparition d'un message de bienvenu)

<https://petstore.octoperf.com/actions/Catalog.action>

Exercices Cucumber

Exercice 1.2

Automatisez les steps de ce feature



Utilisation de Gherkin / Cucumber

La gestion des données – plan de scénario

Cucumber offre la possibilité de synthétiser l'écriture de scénarii en cas de répétitions de données grâce à l'utilisation du mot clefs « Scenario Outline » associé à un tableau

Scenario: Eat 5 out of 12
Given there are 12 cucumbers
When I eat 5 cucumbers
Then I should have 7 cucumbers

Scenario: Eat 5 out of 20
Given there are 20 cucumbers
When I eat 5 cucumbers
Then I should have 15 cucumbers



Scenario Outline: Eating
Given there are <start> cucumbers
When I eat <eat> cucumbers
Then I should have <left> cucumbers

Examples:

start	eat	left
12	5	7
20	5	15

Exercices Cucumber

Exercice 2

1- Créez un nouveau .feature et récrivez votre test sous la forme d'un plan de scénario utilisant les JDD suivants

login	password	Nom utilisateur
j2ee	j2ee	ABC
ACID	ACID	ABC

2- Exécutez le feature que constatez-vous ?

3- Réalisez les modifications nécessaires à la bonne exécution du script



Utilisation de Gherkin / Cucumber

La gestion des données – la table de données

Contrairement, au « plan de scénario », un script comportant une table de données n'est joué qu'une seule fois, quelque soit le nombre de ligne du tableau

```
Feature: achat
Scenario: Verifier la fonctionnalité achat d'un produit
  Given Le prix des produits de cette liste
    | name | prix |
    | café | 1 |
    | donut | 2 |
  When Je commande 2 "café"
  And Je commande 1 "donut"
  Then Je dois payer 4 euros
```

Ces données doivent être manipulées pour ce qu'elles sont : soit...

- Une liste de listes (liste produit + liste prix)
- Des couples clés-valeurs (Map)
- Une liste de couples clés-valeurs
- Un tableau à deux entrées

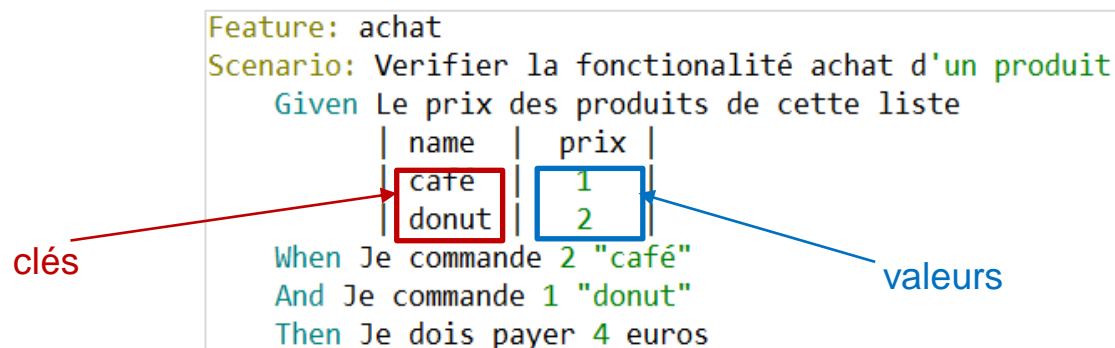
L'automaticien choisira une manière de manipuler ces données selon trois facteurs :

- l'existence d'une ligne d'entête (noms de colonnes)
- le nombre de colonne du tableau
- son appétence personnelle à la manipulation de données sous forme de liste, de table, de map etc..

Utilisation de Gherkin / Cucumber

La gestion des données – la table de données

Ici, on serait tenté de vouloir manipuler ce tableau comme une succession de couple clés (nom du produit) et valeurs (prix du produit)



```
Feature: achat
Scenario: Verifier la fonctionnalité achat d'un produit
  Given Le prix des produits de cette liste
    | name | prix |
    | café | 1    |
    | donut | 2    |
  When Je commande 2 "café"
  And Je commande 1 "donut"
  Then Je dois payer 4 euros
```

Il s'agirait donc d'une **Map<String, Integer>**

```
@Given("Le prix des produits de cette liste")
public void le_prix_des_produits_de_cette_liste(Map<String, Integer> liste_produit) {
  ...
}
```

Mais à l'exécution, un problème apparaîtra, car la java tentera d'interpréter la Map comme suit :

```
{
  "name": prix,
  "café": 1,
  "donut": 2,
}
```

Utilisation de Gherkin / Cucumber

La gestion des données – la table de données

Feature: achat
Scenario: Verifier la fonctionnalité achat d'un produit
Given Le prix des produits de cette liste

nom	prix
café	1
donut	2

When Je commande 2 "café"
And Je commande 1 "donut"
Then Je dois payer 4 euros

Supprimer le nom des colonnes, car toute la liste est considérée comme un couple clé-valeur.

Exemple d'implémentation possible avec une Map :

```
public class AnnotationSteps {  
  
    private Map<String, Integer> liste_produit;  
    int total_cost;  
  
    @Given("Le prix des produits de cette liste")  
    public void le_prix_des_produits_de_cette_liste(Map<String, Integer> liste_produit) {  
        this.liste_produit=liste_produit;  
    }  
  
    @When("Je commande {int} {string}")  
    public void je_commande(Integer nombre_produit, String produit) {  
        int price_produit = liste_produit.get(produit);  
        total_cost += price_produit*nombre_produit;  
    }  
  
    @Then("Je dois payer {int} euros")  
    public void je_dois_payer_euros(int prix_attendu) {  
        assertEquals(prix_attendu, total_cost);  
    }  
}
```

```
Map<String, Integer>  
{  
    "café": 1,  
    "donut": 2,  
}
```

Utilisation de Gherkin / Cucumber

La gestion des données – la table de données

Pour conserver les entêtes de colonnes, il faut manipuler ces données comme une liste d'ensemble de couple clés (nom de colonne) et valeurs (valeur de donnée)

```
Feature: achat
Scenario: Verifier la fonctionnalité achat d'un produit
  Given Le prix des produits de cette liste
    

| name  | prix |
|-------|------|
| café  | 1    |
| donut | 2    |


  When Je commande 2 "café"
  And Je commande 1 "donut"
  Then Je dois payer 4 euros
```

List<Map<String, String>>

```
[
  {"name": "café",
   "prix": "1"},
  {"name": "donut",
   "prix": "2"},
]
```

Map<String, Integer>

```
{
  "café": 1,
  "donut": 2,
}
```

Utilisation de Gherkin / Cucumber

La gestion des données – la table de données

```
Feature: achat
Scenario: Verifier la fonctionnalité d'achat d'un produit
Given le prix de produit de cette liste


| name  | prix |
|-------|------|
| café  | 1    |
| donut | 2    |


When je commande 2 "café"
And je commande 1 "donut"
Then je dois payer 4 euros
```

Exemple d'implémentation possible avec un objet Produit:

```
public class AnnotationSteps {

    private List<Map<String, String>> liste_produit;
    int total_cost;

    @Given("Le prix des produits de cette liste")
    public void le_prix_des_produits_de_cette_liste(List<Map<String, String>> liste_produit) {
        this.liste_produit=liste_produit;
    }

    @When("Je commande {int} {string}")
    public void je_commande(Integer nombre_produit, String produit) {
        for (Map<String,String> m : liste_produit) {
            if(m.get("name").equals(produit)) {
                int price_produit = Integer.parseInt(m.get("prix"));
                total_cost += price_produit*nombre_produit;
            }
        }
    }

    @Then("Je dois payer {int} euros")
    public void je_dois_payer_euros(int prix_attendu) {
        assertEquals(prix_attendu, total_cost);
    }

}
```

```
List<Map<String, String>>
[
  {"name": "café",
   "prix": "1"},
  {"name": "donut",
   "prix": "2"},
]
```

Exercices Cucumber

Exercice 2 (part 1)

A partir de ce *.feature*, implémenter le code et le tester

USER STORY

In order to control casino entrance

As a bouncer

I Want To know who is allowed to enter the casino (age ≥ 18 and no gambler)

Feature: Casino entrance control

Scenario: A groupe of persons who are allowed to enter the casino ask the bouncer

Given the following persons

age	gambler
18	false
26	false
120	false

When they ask if they could go in a casino

Then the bouncer should say "of course, come in"

Exercices Cucumber

Exercice 2 (part 2)

Ajouter ce *Scenario* au .feature de l'exercice 2 et améliorer le code de manière à ce que le test passe en succès.

Scenario: A groupe of persons who are not allowed to enter the casino ask the bouncer

Given the following persons

	age		gambler	
	18		true	
	26		false	
	12		false	

When they ask if they could go in a casino

Then the bouncer should say "no way, get out my face !"

Le Gherkin en Français



Implémenter des scripts Gherkin en Français à partir de Cucumber implique de porter une attention particulière aux spécificités de la langue, lesquelles pourraient confondre le framework d'automatisation.

Voici quelques trucs et astuces :

https://qdusser01.github.io/squashBDD/WEB/co/Encodage_et_caracteres_speciaux.html



Exercices complémentaires

[Voir pages suivantes](#)



Exercices Cucumber

Exercice 1

A partir de ce *.feature*, implémenter le code et le tester

Feature: Addition

Scenario: Add two numbers

Given I have entered 50 into the calculator

And I have entered 70 into the calculator

When I press add

Then the result should be 120 on the screen

Exercices Cucumber

Exercice 2

A partir de ce *.feature*, implémenter le code et le tester

Feature: salary managment

Scenario Outline: Salary management depending on the name of the employee

Given a list of employees salary

When it is a <name>

Then the salary should be <pay>

Examples:

name	pay
bob	35k€
bill	50k€