



# Automatisation des tests fonctionnels avec Cucumber

# Agenda

- Introduction
- Spécification en Gherkin
- Cucumber



# Chap. 1: Introduction

- Contexte
- Les approches TDD
- Les bénéfices de la BDD



# 1.1 Test continu

## Le tests efficace

**Intégrer tous les acteurs dans la démarche de test à toutes les étapes de construction du produit**

**Intégrer l'exécution des tests à toutes les étapes de construction et livraison du produit**



### Tester de la valeur métier

Les tests doivent permettre de représenter les comportements des fonctionnalités attendues afin de tester ce qui est nécessaire et qui apporte de la valeur métier.



### Tester ensemble

La qualité est de la responsabilité de **tous les acteurs du projet**. Ils doivent intervenir pour garantir que chaque couche de construction du produit répond aux attentes et apporte cette valeur.



### Tester tôt

Tester le plus tôt possible permet de prévenir des défauts et de construire une solution robuste. Intégrer les tests tôt dans la construction du projet permet de sécuriser les transformations en cours (Agile, Digitale, Externalisation).



# 1.2 Test-Driven Development

## Développement piloté par les tests

**Test First:** L'approche des développements dirigés par les tests réside dans la spécification des tests avant la réalisation du code.

- **TDD (Test Driven Development)**  
Réalisation en amont des Tests Unitaires par les développeurs
- **BDD (Behavior Driven Development)**  
Réalisation en amont des tests basés sur le comportement du système
- **ATDD (Acceptance Test Driven Development)**  
Réalisation en amont des tests d'acceptance

1. Ecrire les tests

2. Développer

3. Exécuter les tests

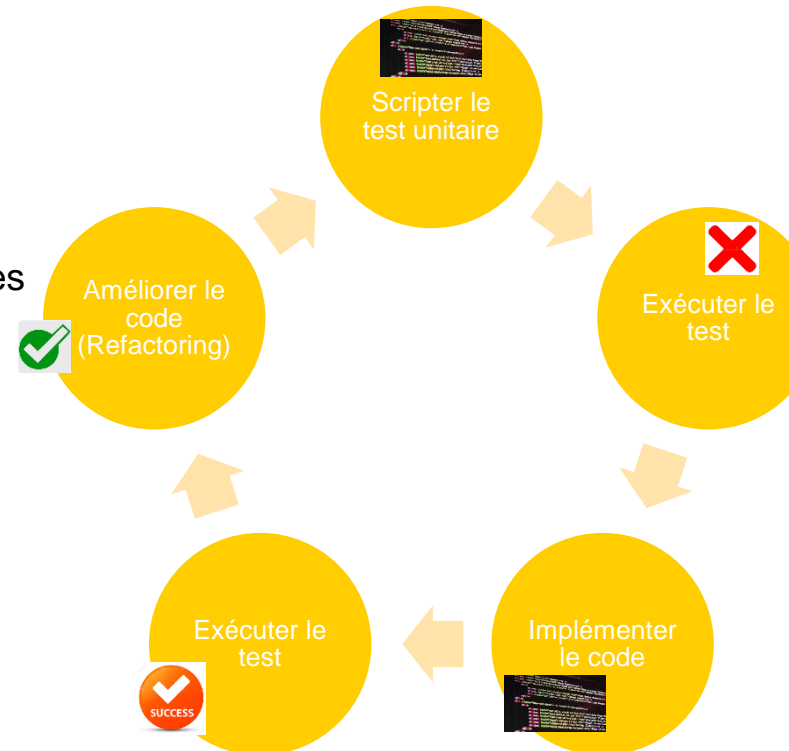


# 1.2 Test-Driven Development

## Ecrire le test avant de coder

- Ecriture des tests unitaires automatisés pour diriger l'écriture du code source

- Ecriture d'un premier test
- Exécuter le test et vérifier qu'il échoue  
Ecriture de l'implémentation pour faire passer le test  
(il existe différentes manières de corriger ce code)
- Exécution des tests afin de contrôler que les tests valident le code et ce dernier respectera les règles fonctionnelles des tests unitaires
- Remaniement (Refactoring) du code afin d'en améliorer la qualité mais en conservant les mêmes fonctionnalités
- Exécution de l'ensemble des tests pour vérifier qu'ils fonctionnent toujours

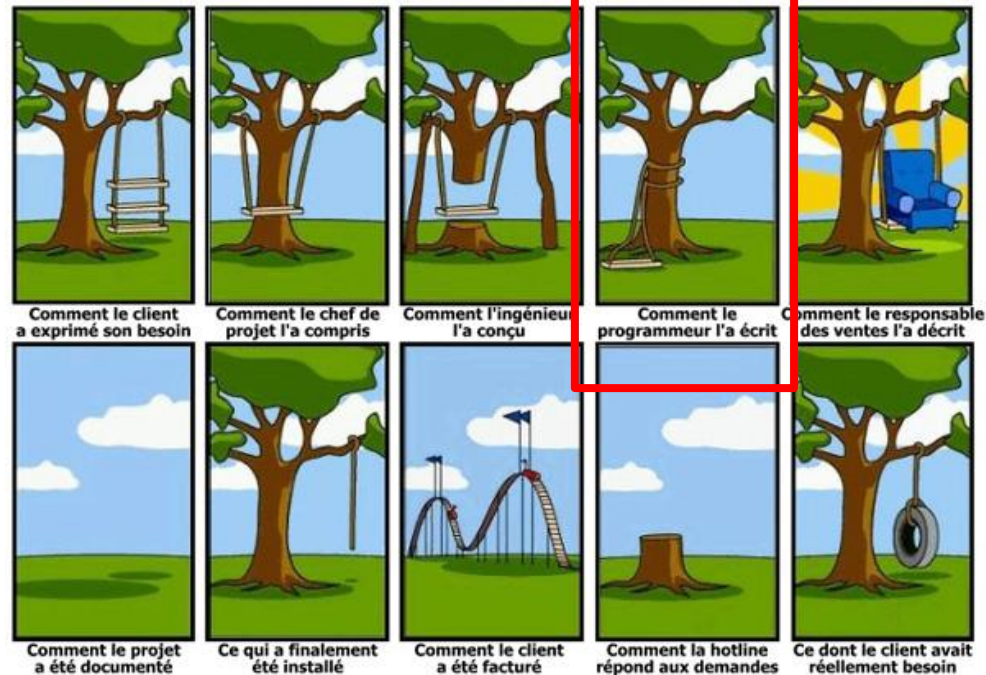


# 1.2 Test-Driven Development

## Une vision unique

**Approche TDD: orientée sur une vision de type boîte blanche**

Unilatérale: seule la connaissance du développeur pour spécifier les tests  
Basée uniquement sur le code



# 1.2 Test-Driven Development

## Behaviour Driven Development: genèse

- **Nommage des méthodes de tests par des phrases**

➔ Documentation des Tests Unitaires

➔ Formalisation en langage métier: Discuter des cas avec les utilisateurs

```
public class CustomerLookupTest extends TestCase {  
    testFindCustomerById() {  
        ...  
    }  
  
    testFailsForDuplicateCustomers() {  
        ...  
    }  
    ....  
}
```



```
CustomerLookup  
- finds customer by id  
- fails for duplicate customers  
- ...
```

- **Comportement**

Test ➔ on se focalise beaucoup sur le résultat attendu

Comportement ➔ on se focalise sur l'objet dans un contexte métier donné





# 1.2 Behaviour Driven Development

**Approche orientée boîte noire:** Ensemble de pratiques de développement favorisant la construction de produits logiciels de bonne qualité en encourageant et impliquant tous les acteurs du produit dans une démarche de tests le plus tôt possible.

- BDD = TDD améliorée
- Basée sur le comportement « réel » attendu du logiciel déduit des besoins métiers.

- **Piloter le code par la valeur métier:**

Identifier l'objectif attendu

Identifier les bénéfices attendus



**On ne teste pas seulement le code**



Validation du comportement de la fonctionnalité dans son contexte d'utilisat



# 1.2 Behaviour Driven Development

Nous aimerions encourager les nouveaux utilisateurs à acheter nos produits.  
Nous proposerons une remise de 10% sur leur première commande.

```
Public void testPremierOrdreNouveauClient()
{
    Client nouveauClient = new Client();
    Commande nouvelleCommande = new Commande(nouveauClient);
    nouvelleCommande.AjoutLire(Catalogue.Trouver("ISBN-0031312121");
    Assert.Equals(10.50, nouvelleCommande.Total);
}
```

Je me connecte avec "jsmith"  
Je clique sur **Catalogue**  
Je saisis "ISBN-0031312121" dans le champ de **Recherche**  
Je clique sur **Ajout au panier**  
Je clique sur **Visualiser le panier**  
Je vérifie que le total est de **10,50 €**



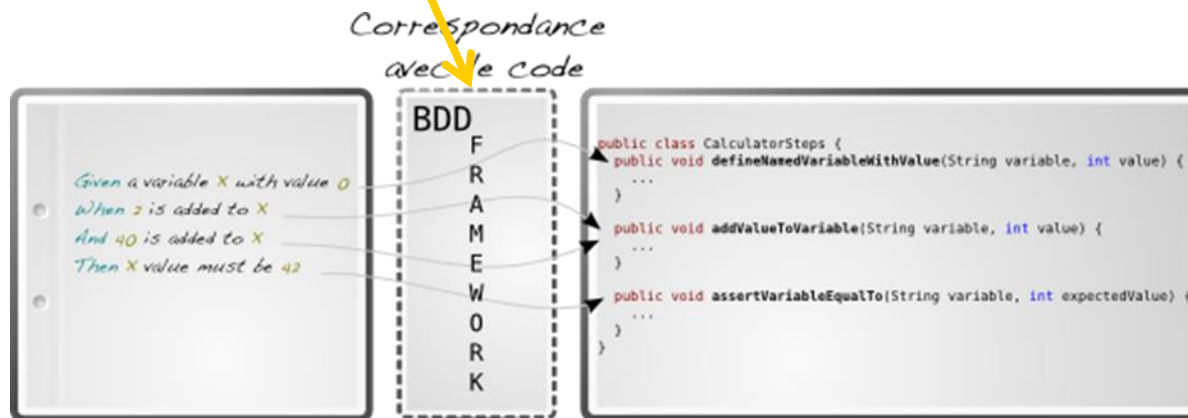
# 1.2 Behaviour Driven Development

- 3 Règles essentielles dans la démarche:

Règle 1: Les tests sont construits de façon collaborative

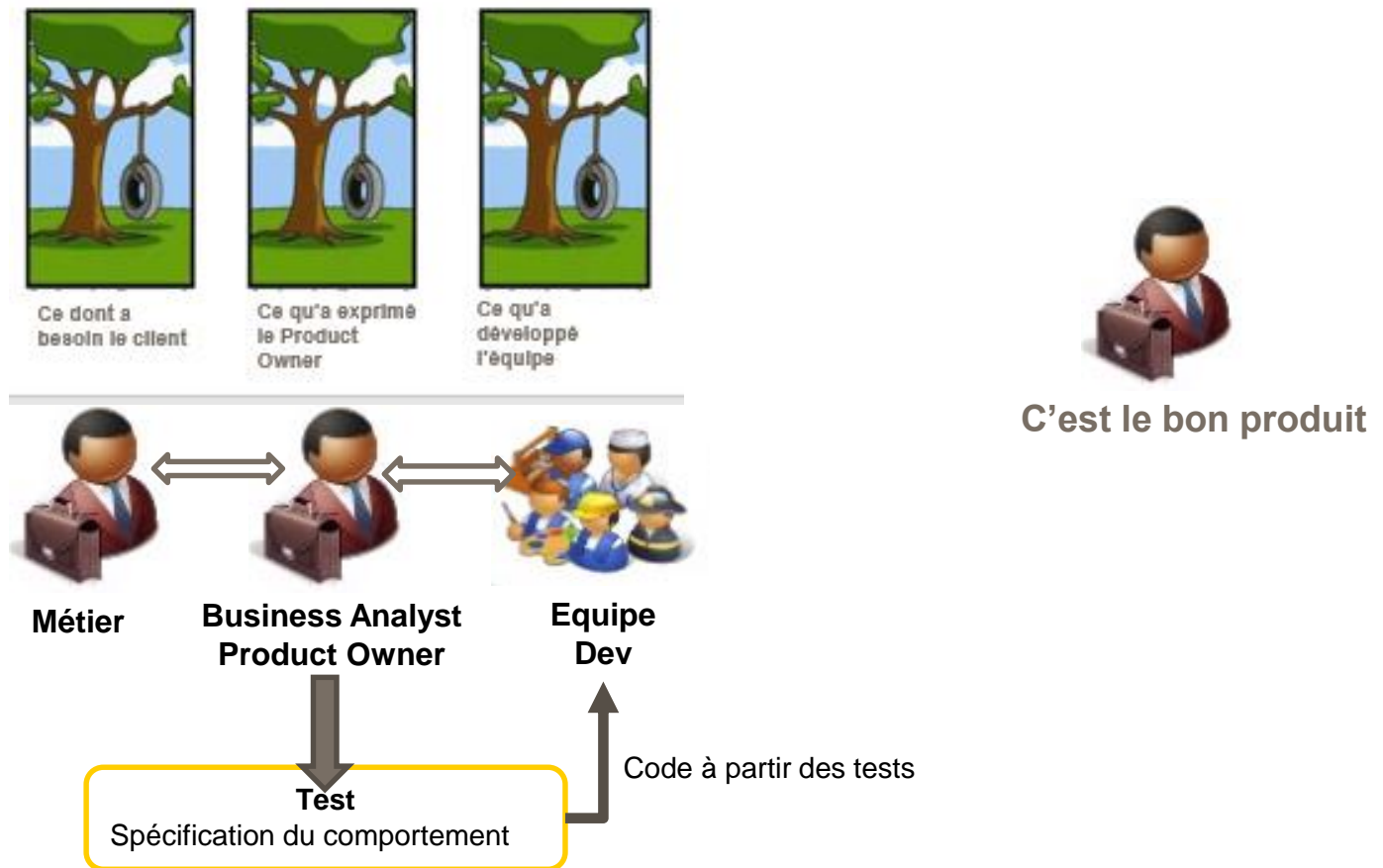
Règle 2: Les tests sont formalisés dans un langage formel (naturel)

Règle 3: Les tests doivent être automatisés le plus tôt possible



# 1.3 Les bénéfices

## La qualité au sein du projet et de l'équipe



# 1.3 Les bénéfices

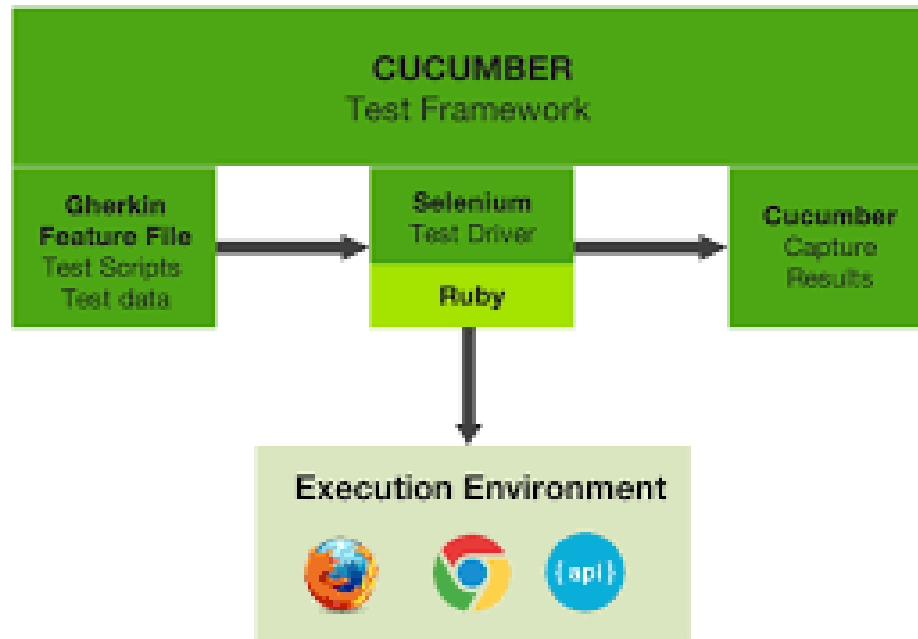
- Favorise la **collaboration des équipes**
- Facilite la réalisation **des spécifications fiables et exécutables** (documentation fonctionnelle)  
Validation des Spécifications du comportement des fonctionnalités
- Clarifie les besoins et **élimine les défauts de conception** (ambiguïté,...)
- Permet une correction rapide des défauts et la validation du comportement de la fonctionnalité
- Aide à la définition de **scénarios de non régression**
- On teste Juste ce qu'il faut



Il est plus facile et confiant de refaire ou modifier un code avec des garde-fous comme les tests BDD et Unitaires.



# 1.3 Outillage BDD



# Chap. 2: Le Gherkin

- Définition
- Ecriture en Gherkin
- Gestion des données



# 2.1 Le Gherkin

## Définition



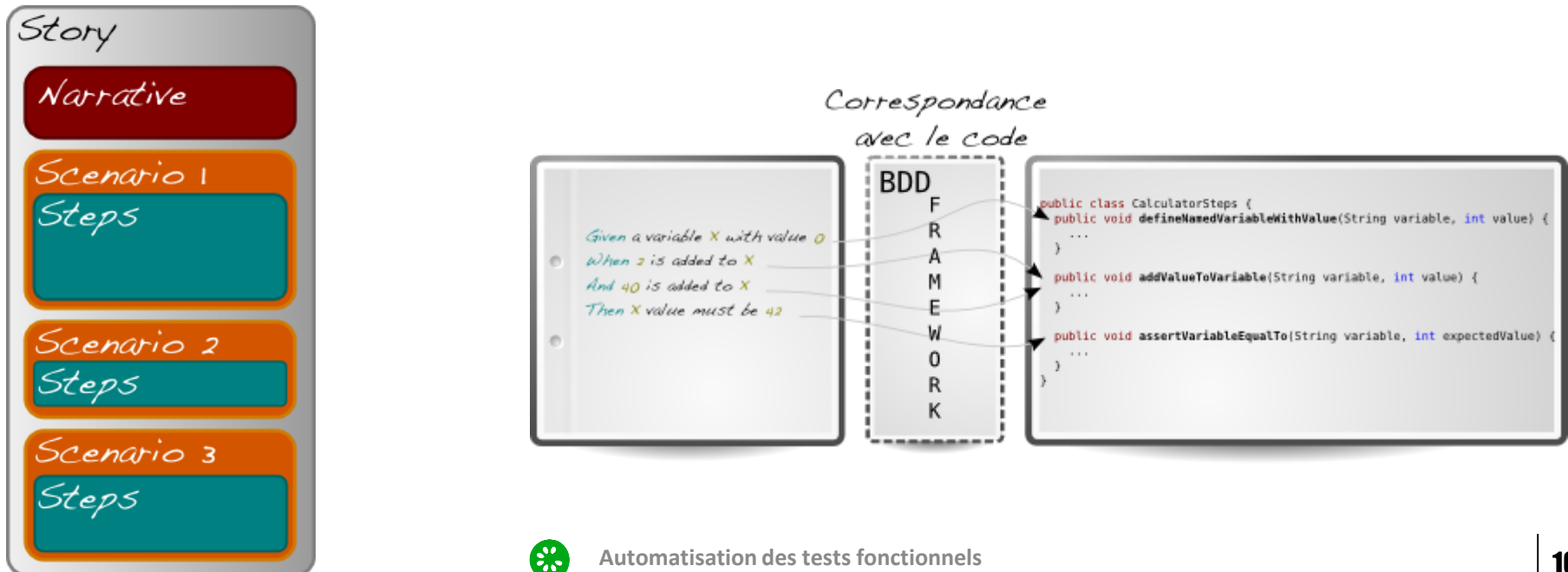
**Outil de formalisation des tests BDD pour décrire les scénarios de tests**

Le Gherkin permet de formaliser dans un langage formel le comportement attendu d'un logiciel.

Ce langage formel est **compris par tous** (développeurs et non technique).

Il permet de formaliser les fonctionnalités du produit à tester.

Il facilite l'automatisation des tests grâce à des frameworks associés pour analyser le langage formel et le traduire en script de test.





# 2.1 Le Gherkin

## La formalisation des scénarios

- Les comportements décrivent la réponse du système d'un point de vue métier
- Les tests, décrivant le comportement du logiciel dans son utilisation cible, doivent être décrits dans un langage formel, unique et compris pour l'ensemble des acteurs du projet.

### Spécification d'un comportement



Conditions  
opérationnelles



Opérations



Résultats



# 2.1 Le Gherkin

## La formalisation des scénarios:

- On définit le comportement fonctionnel attendu par le code:

**Réaction du système lors d'une opération fonctionnelle (métier)**

*Comment réagit mon système quand je me connecte avec des identifiants invalides?*

*Comment réagit mon système quand je saisis une commande sur un produit en stock?*

- Le comportement doit obéir à **une valeur métier**



## 2.1 Le Gherkin: Exemple

**Fonctionnalité:** Servir du café

Afin de gagner de l'argent  
Les clients doivent être capables  
d'acheter du café à toutes heures

**Scénario:** Acheter le dernier café

Etant donné qu'il reste 1 café dans la machine  
Et que j'ai mis 1 dollar  
Quand j'appuie sur le bouton de la machine  
Alors je devrai recevoir un café

**Etant donné** que je dispose de 20 euros sur mon compte bancaire

**Quand** je crédite mon compte de 10 euros

**Alors** mon solde devrait être de 30 euros

Feature: Account Holder withdraws cash

Scenario: Account has sufficient funds

Given the account balance is \$100

And the card is valid

And the machine contains enough money

When the Account Holder requests \$20

Then the ATM should dispense \$20

And the account balance should be \$80

And the card should be returned



## 2.2 Ecrire le Gherkin: Feature

Un fichier Gherkin est une feature: xxxx.feature

Le mot-clé **Feature** est le premier du fichier.

Généralement, la feature représente la fonctionnalité testée

L'objectif de la feature est décrit en formalisme:

**En tant que** [rôle ou personne],

**Je veux/devrais** [fonctionnalité]

**Afin de** [but, bénéfice, valeur]

Il est important de déterminer la finalité métier: **Afin de ....**

@Café

Fonctionnalité: Servir Café

Les Clients devraient pouvoir acheter du café tout le temps

Afin de gagner de l'argent

Fonctionnalité: Calcul du prix du panier

En tant que client connecté,

Je dois pouvoir connaître le prix de mon panier d'achats en cours à tout moment

Afin d'acheter en toute confiance



## 2.2 Ecrire le Gherkin: Scenario

Un scénario correspond à un **comportement du système par rapport** à l'objectif de la feature.

Il décrit un contexte d'exécution de la fonctionnalité.

Une feature peut inclure plusieurs scénarios.

Scénario:

```
Etant donné que je suis sur mon panier
Et que j'ai un produit d'id "1235" en quantité "1"
Et que le stock restant sur ce produit est de "10"
Quand j'ajoute "1" quantité de mon produit
Alors mon produit aura "2" quantités
```

Scénario: Ajout d'articles à mon panier

```
Etant donné je suis un client connecté
Lorsque Je rajoute 2 articles "Iphone X" à mon panier
Alors l'article est mis au panier
Et la quantité d'article a été incrémenté de 2
Et le prix a augmenté de la somme des 2 articles
```

Scénario: Envoi des feuilles de paie par e-mail

```
Tous les mois, les employés reçoivent leurs
feuilles de paie par e-mail.
```

```
Étant donné un employé
```

```
Et nous sommes en avril 2015
```

```
Lorsque l'on passe en mai 2015
```

```
Alors il devrait recevoir un e-mail "Bulletin de paie
avril 2015"
```

```
Et il devrait avoir une pièce jointe "2015-04.pdf"
```



## 2.2 Ecrire le Gherkin: Steps

### **Given:** Contexte (Arrange/Setup)

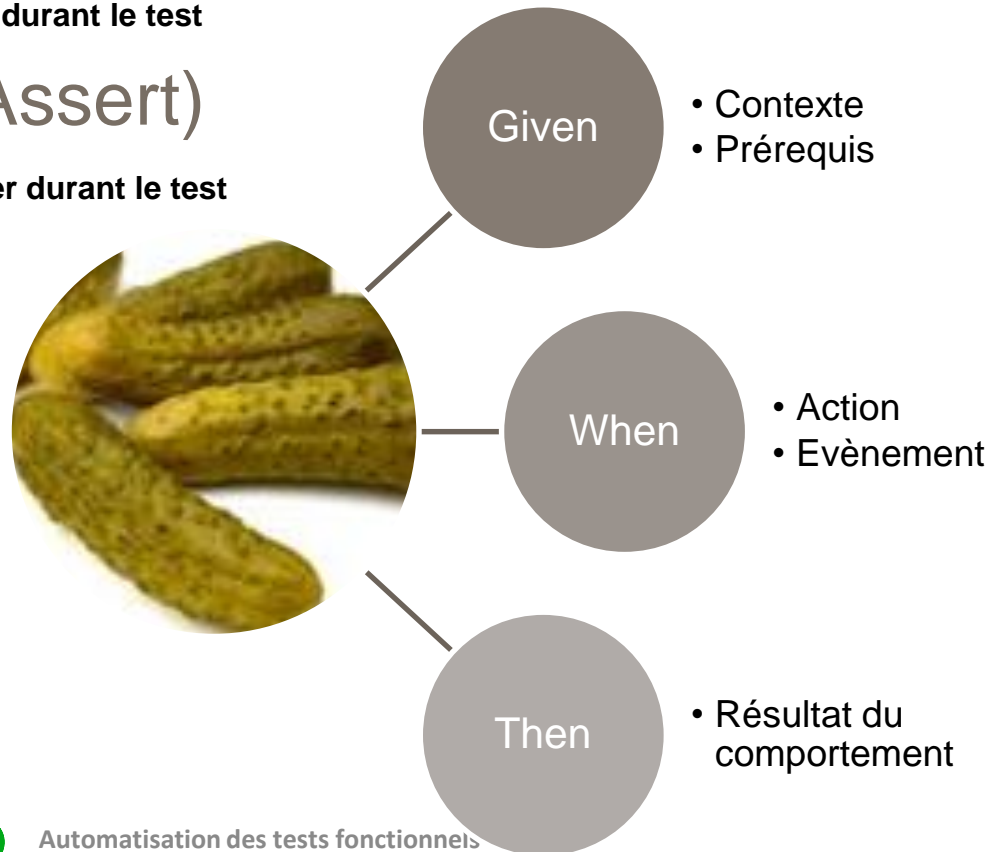
Permet de définir le contexte, les préconditions, l'état initial avant le test

### **When:** Opérations (Act)

Permet de définir les opérations à mener durant le test

### **Then:** Vérification (Assert)

Permet de définir les vérifications à mener durant le test



## 2.2 Ecrire le Gherkin

**Fonctionnalité:** Connexion à l'application

Description: En tant qu'utilisateur non authentifié,  
je dois pouvoir me connecter avec des identifiants valides

**Scénario:** Connexion avec identifiants invalides

**Etant donné** un utilisateur non authentifié

**Quand** Je saisis des identifiants invalides

**Et** Je valide

**Alors** Je n'accède pas à l'application

**Mais** je suis invité à demander un accès



## 2.3 Gestion des données

### Gestion des données: Datatable pour un step

**Permet d'associer une ligne de données à un step**

Scenario Connexion avec identifiants invalides

Etant donné un utilisateur non authentifié

Quand Je saisis mes identifiants

| login | mot de passe |

| bob | mauvaispassword |

Et Je valide

Alors Je n'accède pas à l'application





# 2.3 Gestion des données

## Scenario Outline/Example

**Permet d'associer plusieurs lignes de données à un scénario**

Plan de Scénario: Connexion avec identifiants invalides

Etant donné un utilisateur non authentifié

Quand Je saisis mes identifiants «<login>»: «<mot de passe> »

Et Je valide

Alors Je n'accède pas à l'application

Et le message «<message>»: est affiché

Exemples:

| login | mot de passe | message |  
| bob | mauvaispassword | mot de passe incorrect |  
| bob | | le mot de passe est obligatoire |  
| userexpired | bonpassword | Utilisateur expiré |



# 2.3 Gestion des données

## Background

Le background permet de répéter une même suite de steps avant chaque scénario de la feature.

```
Background: User is Logged In
  Given I navigate to the login page
  When I submit username and password
  Then I should be logged in
```

Contexte:

Etant donné Je lance Dolibarr  
Et je connecté en tant que jsmith

@1-Haute

Scénario: Un commercial devrait pouvoir créer un tiers

Etant donné Je suis un commercial authentifié

Lorsque Je crée un nouveau tiers avec les informations suivantes

|              |                   |             |
|--------------|-------------------|-------------|
| Nom du tiers | Prospect / Client | Fournisseur |
| BPI France   | Client            | Non         |

Alors Le nouveau tiers est activé avec un identifiant

Et Il est possible de créer une proposition commerciale

@1-Haute

Scénario: Un commercial devrait pouvoir créer un tiers à l'international

Etant donné Je suis un commercial authentifié

Lorsque Je crée un nouveau tiers avec les informations suivantes

|              |                   |             |      |
|--------------|-------------------|-------------|------|
| Nom du tiers | Prospect / Client | Fournisseur | Pays |
| BPI France   | Client            | Non         | USA  |

Alors Le nouveau tiers est activé avec un identifiant

Et Il est possible de créer une proposition commerciale

Déroulement du test:

Contexte

Scénario 1

Contexte

Scénario 2

Contexte

....



## 2. TP: Ecrire des scénariis Gherkin



# Chap. 3: Cucumber

- Installation
- Gestion des features
- Automatisation des steps
- Exécution
- Datatable



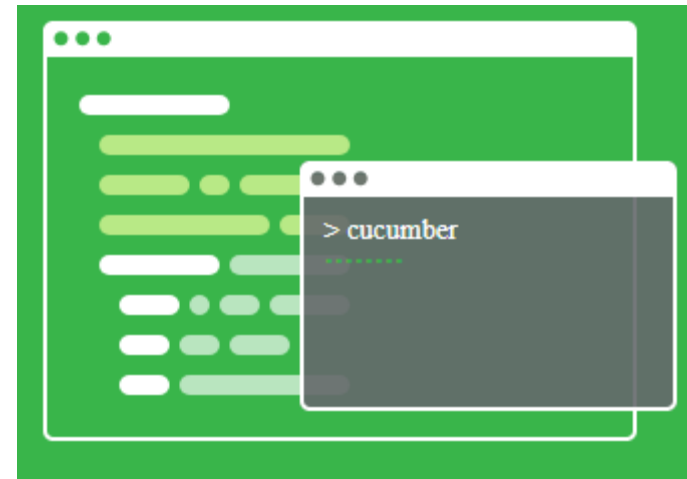
# 3.1 Cucumber

## L'automatisation des tests Gherkin

- Fournit un feedback au plutôt sur la qualité des produits
  - Utilisation de framework pour générer les scripts de tests des scénarios (Spécification Exécutable)
  - ➔ **L'utilisation de framework BDD permet de générer à moindre coût les tests automatiques**
- Fiabilisation des changements lors des développements itératifs

## Cucumber: Framework de tests BDD

- Solution d'automatisation des tests en utilisant l'approche BDD
- Open source
- Multi plateforme: Windows, Linux
- Multi technologie: Java, Ruby, JS, .NET,



# 3.1 Cucumber

## Cucumber-JVM et Plugin

- \* Technologie: Java
- \* Intégration continue: Jenkins

- Méthodologie

- Les scénarios de tests BDD seront gérés dans un projet Java pour l'ensemble de l'application

- Instructions

- Lancer Eclipse
  - Aller dans la MarketPlace
  - Recherche le plugin Cucumber
  - Installer le plugin
  - Redémarrer Eclipse

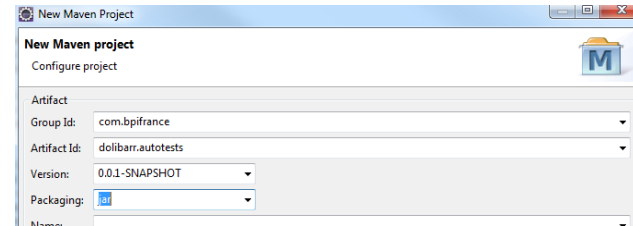


# 3.1 Cucumber

## Java: Création d'un projet Maven

- \* Technologie: Java
- \* Intégration continue: Jenkins

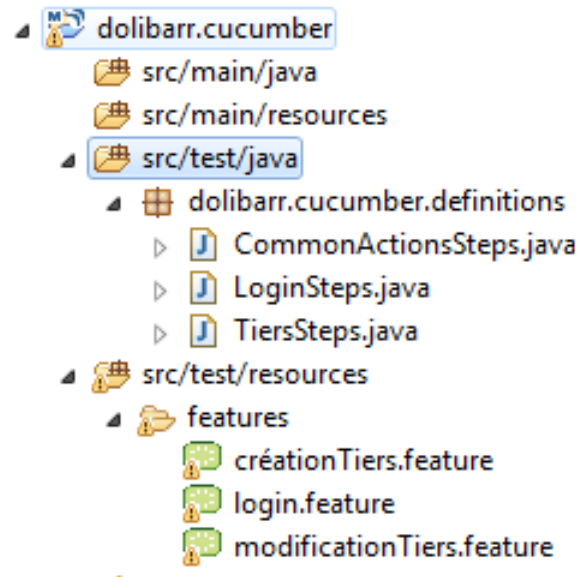
- Créer un projet Maven (archetype simple)
  - Nom du projet: **application**-cucumber
- Rajouter les dépendances suivantes au POM Maven
  - cucumber-java: 1.2.5
  - cucumber-junit: 1.2.5
  - junit: 4.12



# 3.2 Structure

\*

- Structure conseillée
  - Package **definitions**: stocke les scripts des mots-clés
  - Ressources **features**: stocke les différents scénarios Gherkin



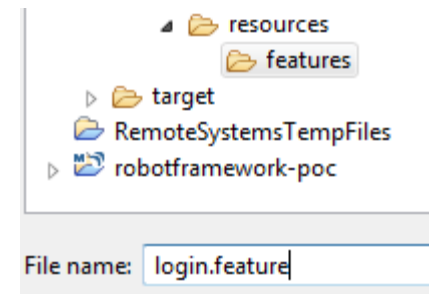


# 3.2 Feature

## Création de la feature Gherkin

Objectif: Créer les scénarios BDD définis par l'équipe

- Aller dans le dossier ressources
- Sélectionner le dossier de la ressource (Organiser par Macro-Fonctionnalité si besoin)
- Créer une feature:
  - Créer un nouveau fichier (File) avec l'extension **.feature**
  - La feature est créée et un template est fourni par le plugin



# 3.2 Feature

Template: Reprend tous les concepts du Gherkin

```
#Author: your.email@your.domain.com
#Keywords Summary :
#Feature: List of scenarios.
#Scenario: Business rule through list of steps with arguments.
#Given: Some precondition step
#When: Some key actions
#Then: To observe outcomes or validation
#And,But: To enumerate more Given,When,Then steps
#Scenario Outline: List of steps for data-driven as an Examples and <placeholder>
#Examples: Container for s table
#Background: List of steps run before each of the scenarios
#""" (Doc Strings)
#| (Data Tables)
#@ (Tags/Labels):To group Scenarios
#<> (placeholder)
#""
## (Comments)
#Sample Feature Definition Template
@tag
» Feature: Title of your feature
  I want to use this template for my feature file

  @tag1
  » Scenario: Title of your scenario
    Given I want to write a step with precondition
    And some other precondition
    When I complete action
    And some other action
    And yet another action
    Then I validate the outcomes
    And check more outcomes

  @tag2
  » Scenario Outline: Title of your scenario outline
    Given I want to write a step with <name>
    When I check for the <value> in step
    Then I verify the <status> in step

  » Examples:
    | name | value | status |
    | name1 | 5 | success |
    | name2 | 7 | Fail |
```



# 3.2 Feature

Aide du plugin: Informe des steps non définies

```
1 #Author: Claude-Henri MARGUERITE
2 #Feature: List of scenarios.
3 # - Test connexion avec des identifiants valides
4 #
5 @login
6 Feature: Connexion
7   En tant qu'utilisateur non authentifié,
8   Je dois pouvoir me connecter avec des identifiants valides
9   Afin de protéger les données de l'application
10
11 @1-Haute
12 Scenario: Test connexion avec des identifiants valides
13   Given Je suis un utilisateur non authentifié
14
15   When Je saisis des identifiants
16     | Login | Mot de passe | Nom complet |
17     | jsmith | dolibarr | John SMITH |
18     | lsmith | dolibarr | Laura SMITH |
19
20   And Je valide
21
22   Then Je suis connecté à Dolibarr
23   And Il est affiché le texte suivant "Espace Accueil"
24
25 @2-Moyenne
26 Scenario Outline: Title of your scenario outline
27   Given Je suis un utilisateur non authentifié
28   When Je saisis des identifiants
29   Then Je ne suis pas connecté
30   And Il est affiché le texte suivant <Message>
31
32 Examples:
33   | Login | Mot de passe | Message |
34   | bob | | Mot de passe obligatoire |
35   | bob | smith | Identifiants login ou mot de passe incorrect |
36   | admin | mauvaismotdepass | Identifiants login ou mot de passe incorrect |
```



# 3.2 Feature

Changement de langue: # language: fr

```
# language: fr
#Author: Claude-Henri MARGUERITE
#Feature: List of scenarios.
# - Test connexion avec des identifiants valides

@login
⇒ Fonctionnalité: Connexion
    En tant qu'utilisateur non authentifié,
    Je dois pouvoir me connecter avec des identifiants valides
    Afin de protéger les données de l'application

⇒ Contexte:
    Etant donné Je lance Dolibarr

    @1-Haute
    ⇒ Scénario: Test connexion avec des identifiants valides
        Etant donné Je suis un utilisateur non authentifié

        Lorsque Je me connecte avec les identifiants suivants
        | Login | Mot de passe |
        | lsmith | dolibarr |
        Alors Je suis connecté à Dolibarr
        Et Il est affiché "Espace Accueil"

    @2-Moyenne
    ⇒ Plan du Scénario: Test connexion avec des identifiants valides
        Etant donné Je suis un utilisateur non authentifié
        Quand Je me connecte avec les identifiants <Login>:<Mot de passe>
        Alors Je ne suis pas connecté à Dolibarr
        Et Il est affiché "<Message>"

    ⇒ Exemples:
        | Login | Mot de passe | Message |
        | bob | | Mot de passe obligatoire |
        | bob | smith | Identifiants login ou mot de passe incorrect |
        | admin | mauvaismotdepass | Identifiants login ou mot de passe incorrect |
```

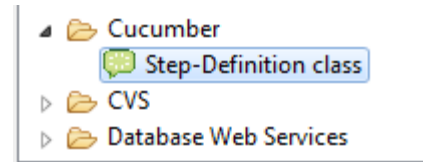


# 3.3 Steps

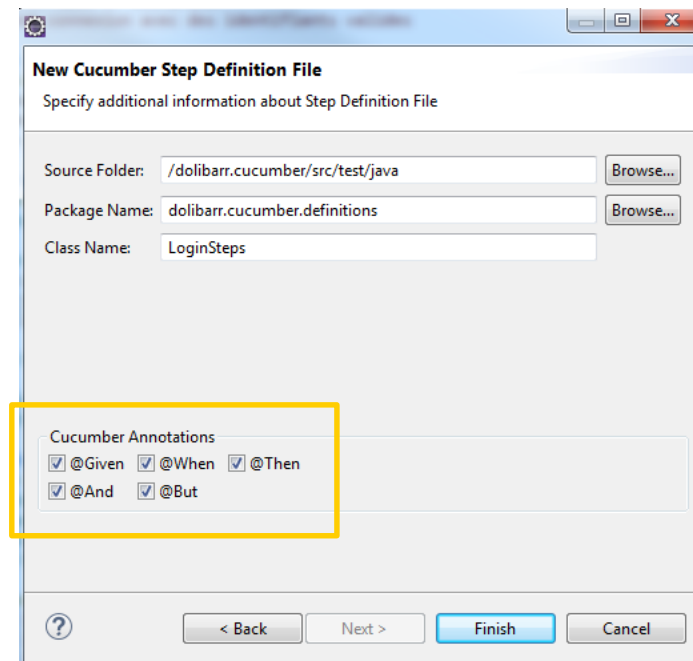
## Création de la classe de Steps Definition

Objectif: Scripter les steps

- Aller dans le dossier package définitions
- Sélectionner le dossier de la ressources (Organiser par Macro-Fonctionnalité si besoin)
- Créer une nouvelle classe de type:



Annotations à intégrer dans la classe



# 3.3 Steps

Librairie Cucumber

```
1 package dolibarr.cucumber.definitions;
2
3 import cucumber.api.java.en.Given;
4 import cucumber.api.java.en.When;
5 import cucumber.api.java.en.Then;
6 import cucumber.api.java.en.And;
7 import cucumber.api.java.en.But;
8
9 public class LoginSteps {
10     @Given("^you are in Given annotation$")
11     public void given() throws Throwable {
12     }
13
14     @When("^you are in When annotation$")
15     public void when() throws Throwable {
16     }
17
18     @Then("^you are in Then annotation$")
19     public void then() throws Throwable {
20     }
21
22     @And("^you are in And annotation$")
23     public void and() throws Throwable {
24     }
25
26     @But("^you are in But annotation$")
27     public void but() throws Throwable {
28     }
29
30 }
```

```
import cucumber.api.java.fr.Etantdonné;
import cucumber.api.java.fr.Lorsque;
import cucumber.api.java.fr.Quand;
import cucumber.api.java.fr.Alors;
import cucumber.api.java.fr.Et;
import cucumber.api.java.fr.Mais;
```

```
public class LoginSteps {
    @Etantdonné("^Je lance Dolibarr$")
```

Annotation Cucumber



## 3.3 Steps

### Association d'une Phrase à un Step

Etant donné Je lance Dolibarr


Cucumber analyse les phrases de la feature et va l'associer avec LA méthode annotée avec l'expression régulière validant la phrase.

```
@Etantdonné("^Je lance Dolibarr$")
public void je_lance_dolibarr() throws Throwable {
```

Une Annotation par phrase

Etant donné Je lance Dolibarr  
Lorsque Je lance Dolibarr  
Et Je lance Dolibarr

```
@Etantdonné("^Je lance Dolibarr$")
public void je_lance_dolibarr() throws Throwable {
    System.out.println("J'ouvre le navigateur");
    System.out.println("Je me connecte à l'adresse de Dolibarr");
}
```



# 3.3 Steps

## Arguments et données

Given Je lance le navigateur IE



```
@Given("^Je lance le navigateur (.*?)$")  
public void je_lance_navigateur(String browser) throws Throwable {  
    System.out.println("Je lance le super navigateur " + browser);  
}
```

Arg1: texte

Arg2: champ

```
@When("^Je saisis (.*?) dans le champ (.*?)$")  
public void je_saisis_texte_dans_le_champs(String texte, String champ) throws Throwable {  
}
```

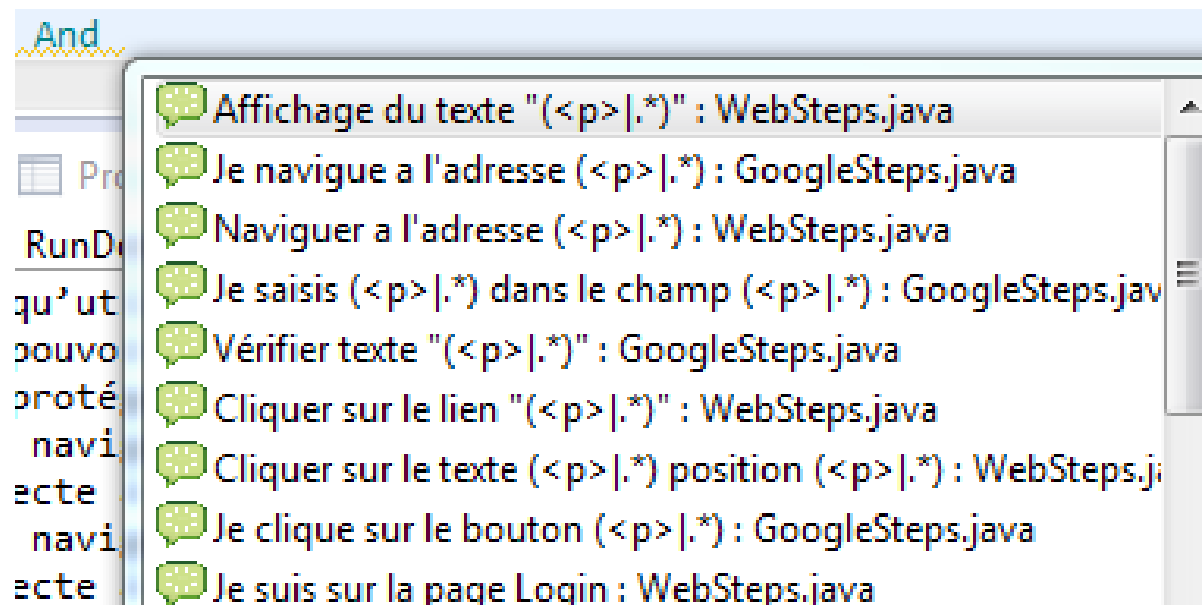




## 3.3 Steps

### Autocompletion de phrases

Le plugin permet de récupérer les steps des définitions pour la saisie d'une feature



# 3.4 Exécution

## Cucumber Runners

Il existe plusieurs façon de lancer le test d'une feature:

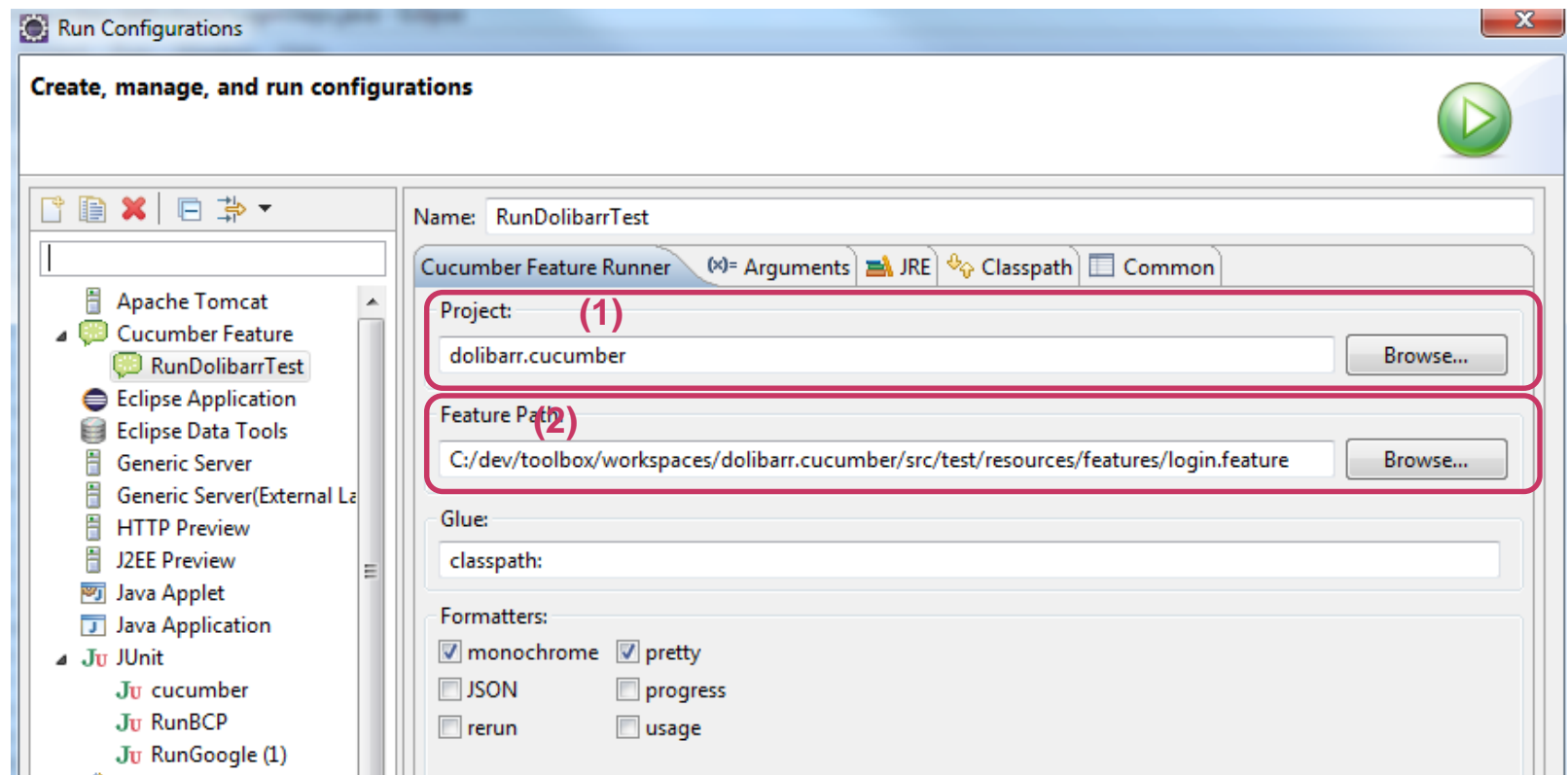
- Avec le runner du plugin
- Avec le client jar cucumber
- Avec un Runner Junit
- Avec maven



# 3.4 Exécution

## Plugin Cucumber

- Fournit par le plugin pour tester rapidement les features
  - (1) Chemin où se trouve les features à exécuter
  - (2) Chemin des classes de définitions



# 3.4 Exécution

## Client jar Cucumber

● `java -cp "classpath des librairies" cucumber.api.cli.Main --glue dolibarr.cucumber.definitions login.feature`

- (1) Chemin des jar (cucumber, projet)
- (2) package des définitions
- (3) Chemin de la feature

```
Usage: java cucumber.api.cli.Main [options] [[[FILE|DIR][:LINE[:LINE]*] ]+ | @FILE ]

Options:

  -g, --glue PATH                Where glue code (step definitions, hooks
                                and plugins) are loaded from.
  -p, --plugin PLUGIN[:PATH_OR_URL] Register a plugin.
                                Built-in formatter PLUGIN types: junit,
                                html, pretty, progress, json, usage, rerun,
                                testing. Built-in summary PLUGIN types:
                                default_summary, null_summary. PLUGIN can
                                also be a fully qualified class name, allowing
                                registration of 3rd party plugins.
                                Deprecated. Use --plugin instead.
  -f, --format FORMAT[:PATH_OR_URL] Only run scenarios tagged with tags matching
  -t, --tags TAG_EXPRESSION       TAG_EXPRESSION.
  -n, --name REGEXP               Only run scenarios whose names match REGEXP.
  -d, --[no-]-dry-run             Skip execution of glue code.
  -m, --[no-]-monochrome         Don't colour terminal output.
  -s, --[no-]-strict             Treat undefined and pending steps as errors.
                                --snippets [underscore|camelcase] Naming convention for generated snippets.
                                Defaults to underscore.
  -v, --version                  Print version.
  -h, --help                     You're looking at it.
  --i18n LANG                   List keywords for in a particular language
                                Run with "--i18n help" to see all languages

Feature path examples:
  <path>                        Load the files with the extension ".feature"
                                for the directory <path>
                                and its sub directories.
  <path>/<name>.feature          Load the feature file <path>/<name>.feature
                                from the file system.
  classpath:<path>/<name>.feature Load the feature file <path>/<name>.feature
                                from the classpath.
  <path>/<name>.feature:3:9      Load the scenarios on line 3 and line 9 in
                                the file <path>/<name>.feature.
  @<path>/<file>                Parse <path>/<file> for feature paths generated
                                by the rerun formatter.
```



## 3.4 Exécution



### Combine le framework de test avec Cucumber

- Junit est connu par les équipes de développement
- Usage des assertions de Junit
- Usage du reporting de test de Junit
- Facilite l'intégration avec les plateformes IC



# 3.4 Exécution



## ● CucumberOptions

- **tags:** permet de choisir les scénarios à exécuter selon leurs tags

tags = {« @NonRegression »} : Exécution des scénarios taggués NonRegression

tags = {« @NonRegression, @Login »} : Exécution des scénarios taggués NonRegression **OU** Login

tags = {« @NonRegression », « @Login »} : Exécution des scénarios taggués NonRegression **ET** Login

tags = {« @NonRegression », « ~@Login »} : Exécution des scénarios taggués NonRegression **en ignorant** Login

- **features:** emplacement des features à analyser pour le Run
- **glue:** emplacement des classes de définition des steps
- **plugin:** spécifie le format de reporting d'exécution
- **monochrome:** rend plus lisible le résultat du test dans la console
- **dryRun:** permet de tester si les phrases ont des steps
- **strict:** arrêt du test si un step n'est pas défini

- Création d'une classe pour Exécuter les tests

```
package dolibarr.cucumber.runners;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
    tags = {"@login"},
    plugin = {"pretty", "html:target/cucumber", "json:target/cucumber.json"},
    dryRun=false,
    strict=true,
    monochrome=true,
    features="src/test/resources/features/",
    glue = {"dolibarr.cucumber.definitions"})

public class RunNonRegression {
}
```



## 3.4 Exécution

The screenshot shows an IDE window with a toolbar at the top containing icons for Markers, Properties, Servers, Data Source Explorer, Snippets, Console, Annotations, and JUnit. Below the toolbar, a status bar indicates "Finished after 0,311 seconds". A progress bar shows "Runs: 20/20", "Errors: 0", and "Failures: 0". The main area displays a tree view of test results for "dolibarr.cucumber.runners.RunNonRegression [Runner: JUnit 4] (0,035 s)". The tree is expanded to show the following structure:

- ▲ Fonctionnalité: Connexion (0,035 s)
  - ▲ Scénario: Test connexion avec des identifiants valides (0,000 s)
    - Etant donné Je lance Dolibarr (0,000 s)
    - Etant donné Je suis un utilisateur non authentifié (0,000 s)
    - Lorsque Je me connecte avec les identifiants suivants (0,000 s)
    - Alors Je suis connecté à Dolibarr (0,000 s)
    - Et Il est affiché "Espace Accueil" (0,000 s)
  - ▲ Plan du Scénario: Test connexion avec des identifiants invalides (0,026 s)
    - ▲ Exemples: (0,026 s)
      - ▶ bsmith | | Mot passe obligatoire | (0,000 s)
      - ▶ bsmith | mauvaismdp | identifiant login ou mot de passe incorrect | (0,001 s)
      - ▶ mauvaislogin | mauvaismdp | identifiant login ou mot de passe incorrect | (0,0

A "Failure Trace" button is visible on the right side of the tree view.



# 3. Exercice: Automatisation





# 3.5 Databale

Phrase avec une table de données en argument

Lorsque Je me connecte avec les identifiants suivants

|        |              |
|--------|--------------|
| Login  | Mot de passe |
| lsmith | dolibarr     |

DataTable

```
@Quand("^Je me connecte avec les identifiants suivants$")  
public void je me connecte avec les identifiants suivants(DataTable data) throws Throwable {  
    List<List<String>> identifiants = data.raw();  
    (2) System.out.println("Je saisis Login: " + identifiants.get(1).get(0));  
    System.out.println("Je saisi Mot de passe: " + identifiants.get(1).get(1)); (3)  
    System.out.println("Je valide");  
}
```

● Dans la méthode:

- (1) On récupère l'objet DataTable
- (2) On crée une liste de liste pour parcourir le tableau Ou une Map
- (3) On récupère la cellule du tableaux

```
@Quand("^Je me connecte avec les identifiants suivants$")  
public void je me connecte avec les identifiants suivants(DataTable identifiants) throws Throwable {  
    for (Map<String, String> data : identifiants.asMaps(String.class, String.class)) {  
        loginPage.setLogin(data.get("Login"));  
        loginPage.setPassword(data.get("Mot de passe"));  
        loginPage.validate();  
    }  
}
```



# 3.5 Datable

Quand je planifie la formation suivante

| code | libelle           | date       | salle | formateur | statut   | ouverte |
|------|-------------------|------------|-------|-----------|----------|---------|
| CSLM | Cucumber Selenium | 18/06/2018 | RDC1  | CHM       | nouvelle | Oui     |

```
public class Formation {  
    private long id;  
    private String code;  
    private String libelle;  
    private Date date;  
    private String salle;  
    private String formateur;  
    private String statut;  
    private int maximumInscrits=10;  
    private int nombreInscrits=0;  
    private boolean ouverte=true;  
    private List<Stagiaire> stagiaires;
```

DataTable

Converter

```
@Quand("^je planifie la formation suivante$")  
public void je_planifie_la_formation_suivante(List<Formation> formations) throws Throwable {  
    formation = formationMetier.planifierFormation(formations.get(0));  
}
```



# 3.5 Databale

## Scenario outline

### Scenario avec une table de données

Plan du Scénario: Test connexion avec des identifiants invalides  
Etant donné Je suis un utilisateur non authentifié

Lorsque Je me connecte avec les identifiants <login>:<mot de passe>  
Alors Je suis un utilisateur non authentifié  
Et Il est affiché "<message>"

Exemples:

| login        | mot de passe | message                                     |
|--------------|--------------|---|
| bsmith       |              | Mot passe obligatoire                       |
| bsmith       | mauvaismdp   | identifiant login ou mot de passe incorrect |
| mauvaislogin | mauvaismdp   | identifiant login ou mot de passe incorrect |

```
@Quand("^Je me connecte avec les identifiants (.*):(.*)$")
public void je_me_connecte_avec_les_identifiants(String login, String mot_de_passe) throws Throwable {

    System.out.println("Je saisis Login: " + login);
    System.out.println("Je saisi Mot de passe: " + mot_de_passe);
    System.out.println("Je valide");
}
```

```
@Alors("^Il est affiché \"(.*)\"$")
public void il_est_affiché(String message) throws Throwable {
    System.out.println("Le texte " + message + " est affiché.");
}
```



# 3.5 Contexte Hook

Permet de rajouter des opérations Avant et Après l'exécution d'un test

```
public class hookTest {
    WebDriver driver = null;

    @Before public void setUp(){
        driver = new FirefoxDriver();
    }

    @Given("^user navigates to facebook$")
    public void goToFacebook() {
        driver.navigate().to("https://www.facebook.com/");
    }

    @When("^I enter Username as \"([^\"]*)\" and Password as \"([^\"]*)\"$")
    public void I_enter_Username_as_and_Password_as(String arg1, String arg2) {
        driver.findElement(By.id("email")).sendKeys(arg1);
        driver.findElement(By.id("pass")).sendKeys(arg2);
        driver.findElement(By.id("u_0_v")).click();
    }

    @Then("^login should be unsuccessful$")
    public void validateReLogin() {
        if(driver.getCurrentUrl().equalsIgnoreCase(
            "https://www.facebook.com/login.php?login_attempt=1&lwv=110")){
            System.out.println("Test Pass");
        } else {
            System.out.println("Test Failed");
        }
        driver.close();
    }

    @After public void cleanUp(){
        driver.close();
    }
}
```

Hook spécifique à un tag

```
@Before("@login")
public void openBrowser()
```



## 4. TP: Cucumber & Selenium

