



Informatique

iut Nord Franche-Comté

TEST D'ACCEPTATION EN JAVA



Informatique
iut Nord Franche-Comté

UNIVERSITÉ DE
FRANCHE-COMTÉ

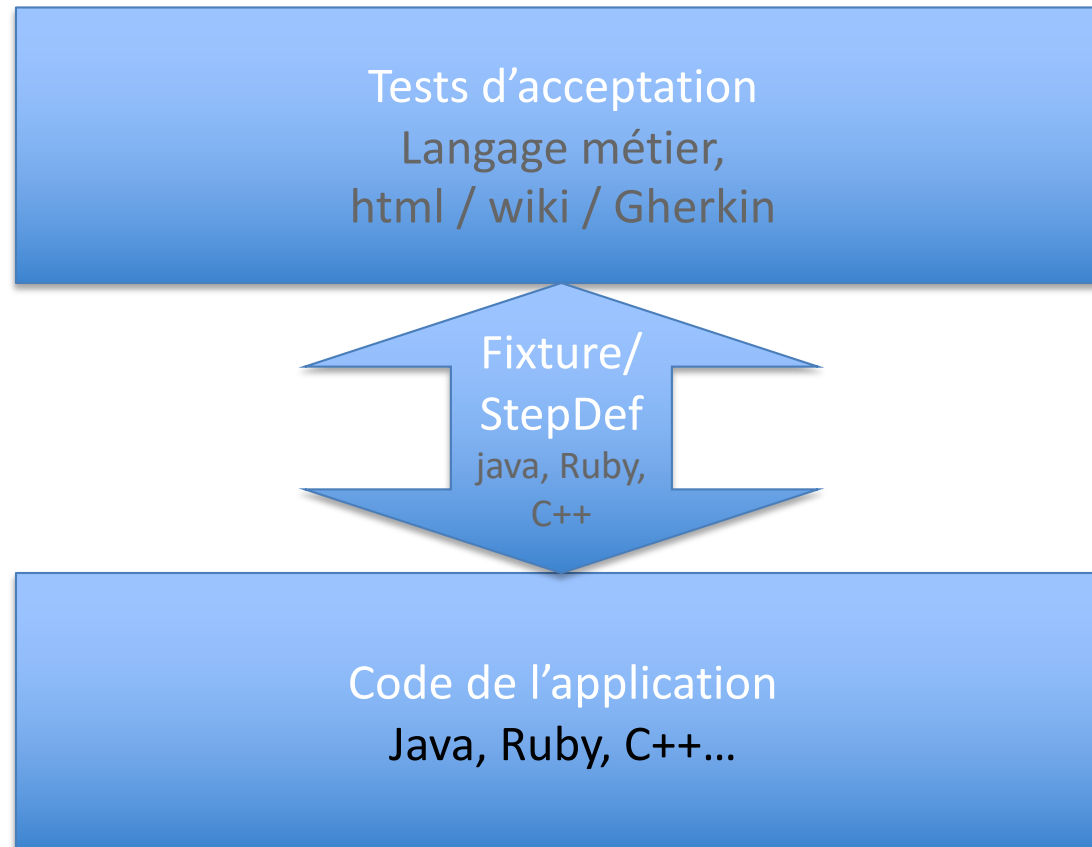
Définition

Un test d'acceptation est un test métier permettant de valider tout ou partie d'une fonctionnalité.

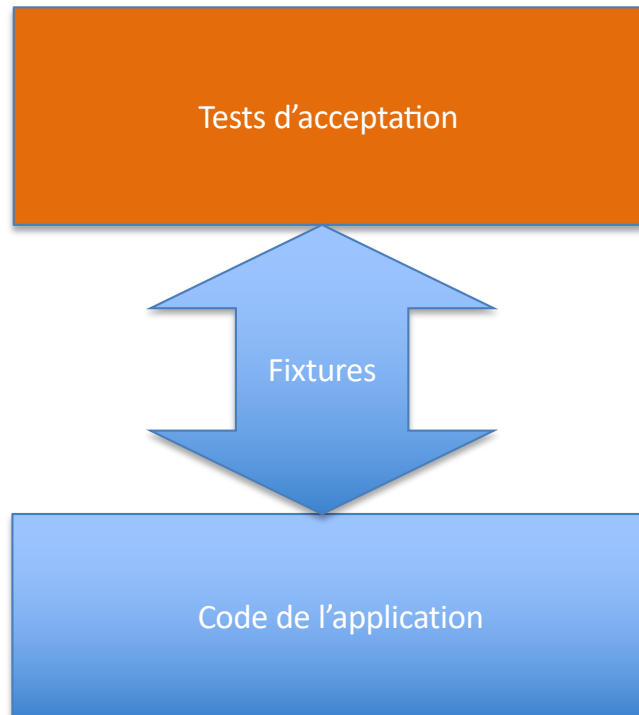
Les tests d'acceptation permettent au client de vérifier qu'une fonctionnalité a été implémentée. Si l'ensemble des tests d'acceptation d'une fonctionnalité sont verts, le client peut accepter la fonctionnalité.

Par nature ce sont des **tests fonctionnels**.

Test d'acceptation & Application



Acteurs du test d'acceptation



Le **client** définit la fonctionnalité à implémenter et les tests d'acceptation associés

Le **développeur** code l'application et les fixtures permettant de réaliser le lien entre les tests d'acceptation et le code

Agilité et Tests d'Acceptation

Les méthodes agiles utilisent des cycles de développement courts pendant lesquels sont pris en charge la réalisation de "stories". La définition et la "mise en page" des tests d'acceptation prennent naturellement place avant de débiter l'implémentation relative à une story.

ATDD : Acceptance Test Driven Development

Outils du Test d'Acceptation

outil			
interface saisie	wiki - tables	html - div Example	texte - Gherkin
interprétation	serveur dédié	Junit	Junit
résultats	dans le wiki	pages html résultat	pages html résultat
http:// https://	fitnesse.org/	www.concordion.org/	cucumber.io

Cucumber is for Behaviour-Driven Development

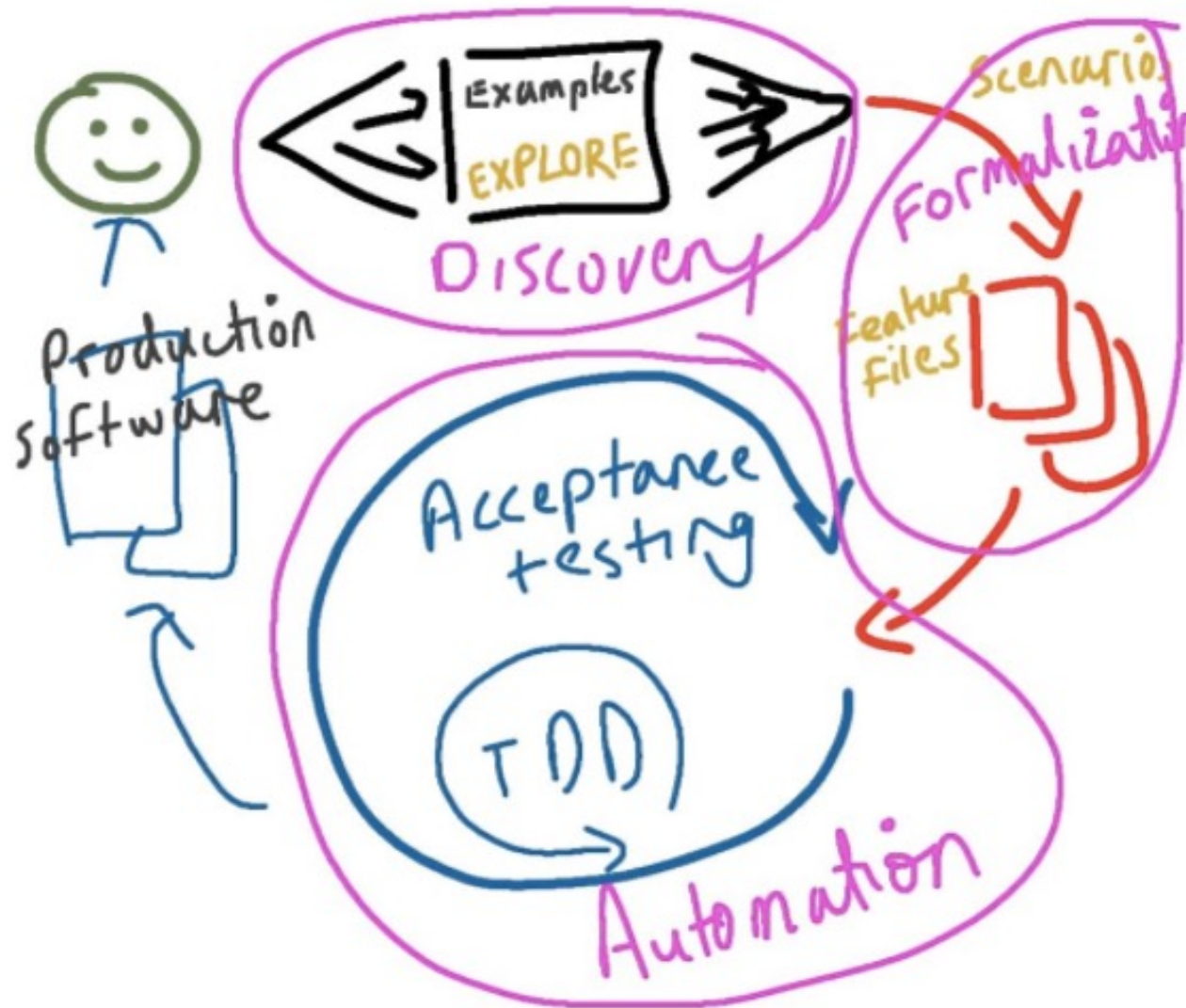


Illustration by Paul Rayner (Thanks!)

image issue de cucumber.io

Utilisation de Cucumber

Distribution cucumber-JVM

```
<dependencies>
  <dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>${cucumber.version}</version>
  </dependency>

  <dependency>
    <groupId> io.cucumber</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>${cucumber.version}</version>
  </dependency>
```


Utilisation de Cucumber

Principes de fonctionnement

- Écrire les fichiers xx.feature en langage Gherkin
 - description textuelle des fonctionnalités
 - utilise les mots clés Given-When-Then
 - disponible dans de nombreuses langues
 - permet de définir des scénarios de test
- Écrire les classes *StepDef* java qui vont :
 - correspondre à chaque actions des scénarios des fichiers feature
 - créer les instances de SUT utiles à l'interprétation du scénario
 - être exécutées pour établir le verdict et produire les rapports

Gherkin : description des fonctionnalités

- Le fichier contient la description de la fonctionnalité
 - sous forme de texte descriptif d'une part
 - au travers de cas d'utilisations donnant des illustrations de la fonctionnalité
- Plusieurs fichiers pour décrire toutes les fonctionnalités
- Les fichiers servent de support à la discussion avec le client
- Les fichiers servent de spécification pour le développement

Gherkin : description des fonctionnalités

dans un fichier texte avec l'extension .feature

Gherkin est disponible dans de nombreuses langues

Feature: Robot

Texte libre permettant de faire la description générale de la fonctionnalité avant l'écriture des scénarios.

language: fr

Fonctionnalité: Robot

Texte libre permettant de faire la description générale de la fonctionnalité avant l'écriture des scénarios.

Scenario: Atterrissage

Given Le robot est en vol

When le robot atterrit en coordonnée (6, 5)

Then les coordonnées du robot sont (6, 5)

And il est orienté vers le nord

Scénario: Atterrissage

Etant donné Le robot est en vol

Quand le robot atterrit en coordonnée (6, 5)

Alors les coordonnées du robot sont (6, 5)

Et il est orienté vers le nord

Feature: Consommation énergie

Texte libre permettant de faire la description générale de la fonctionnalité avant l'écriture des scénarios.

Scenario Outline: influence de la nature de terrain

Given Le robot est posé sur <nature>

And il est orienté vers <direction>

When le robot tourne sur lui-même à gauche

Then il consomme <energy> unité d'énergie

And il est orienté vers <newDirection>

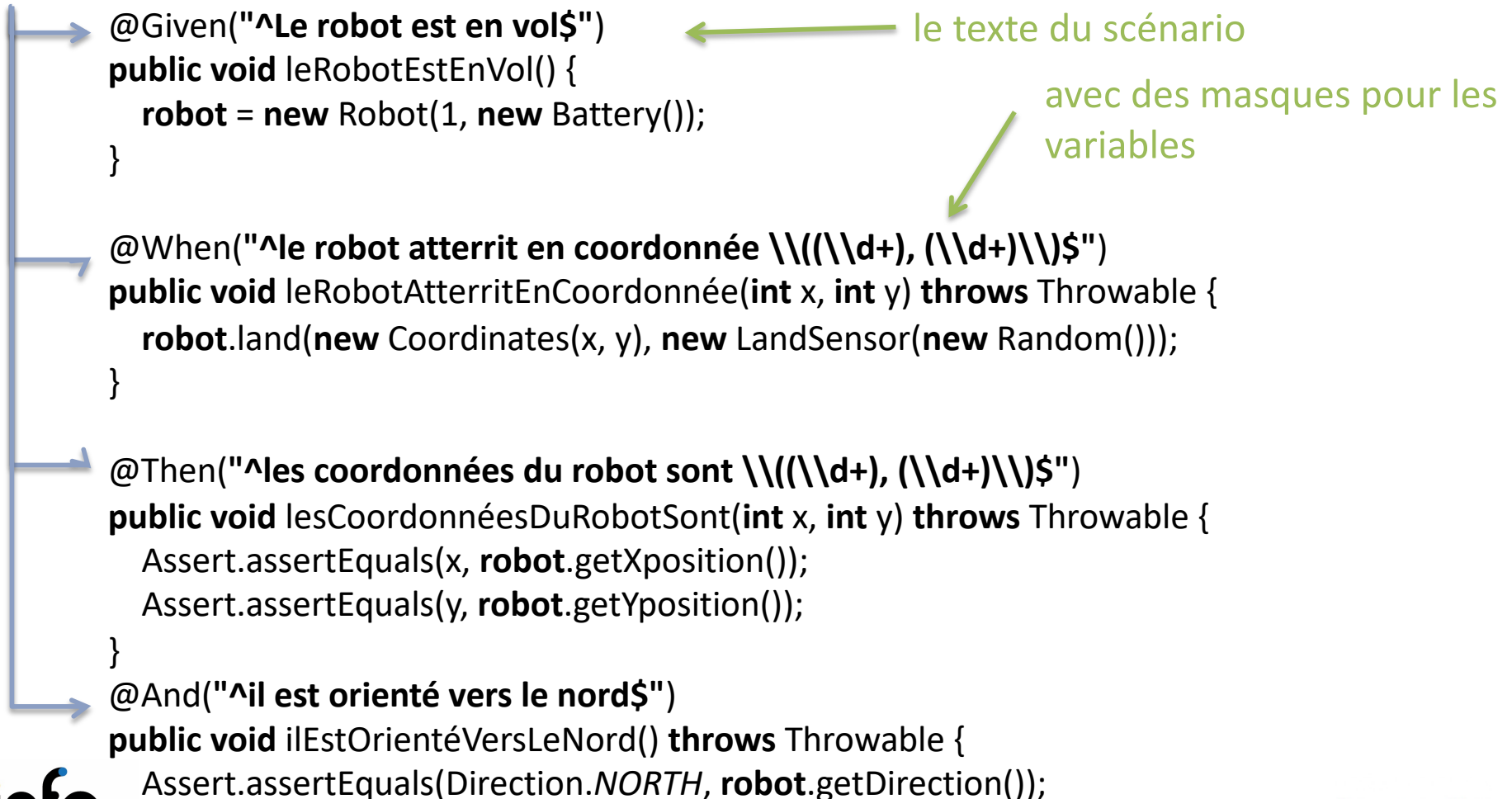
Examples:

nature	direction	energy	newDirection
terre	nord	1	ouest
sable	sud	3	est
boue	est	4	nord
roche	ouest	2	sud

Définition des pas de scénario

Dans une classe java, une méthode par ligne du scénario

annotations spécifiques



```
@Given("^Le robot est en vol$")
public void leRobotEstEnVol() {
    robot = new Robot(1, new Battery());
}

@When("^le robot atterrit en coordonnée \\((\\d+), (\\d+)\\)$")
public void leRobotAtterritEnCoordonnée(int x, int y) throws Throwable {
    robot.land(new Coordinates(x, y), new LandSensor(new Random()));
}

@Then("^les coordonnées du robot sont \\((\\d+), (\\d+)\\)$")
public void lesCoordonnéesDuRobotSont(int x, int y) throws Throwable {
    Assert.assertEquals(x, robot.getXposition());
    Assert.assertEquals(y, robot.getYposition());
}

@And("^il est orienté vers le nord$")
public void ilEstOrientéVersLeNord() throws Throwable {
    Assert.assertEquals(Direction.NORTH, robot.getDirection());
}
```

Exécuter les tests

Cucumber utilise une classe vide

```
package mypackage;
```

```
import cucumber.api.junit.Cucumber;
```

```
import org.junit.runner.RunWith;
```

```
@RunWith(Cucumber.class)
```

```
@CucumberOptions(.....)
```

```
public class RunCukesTest {
```

```
}
```

@CucumberOption

quelques options

- plugin : définition des formats des rapports de test
 - pretty : affichage du source Gherkin
 - html : rapport html dans le répertoire spécifié
 - json : sources gherkin au format json dans le fichier
 - junit : fichier xml du résultat des tests
- feature : localisation des .feature
- glue : localisation des .java

```
@CucumberOptions( plugin = {  
    "pretty",  
    "html:Répertoire",  
    "json:Répertoire/fichier.json",  
    "junit:Répertoire/fichier.xml"  
},  
feature = "Répertoire",  
glue = "Répertoire"  
)
```

Organisation des méthodes

Les pas d'exécution nécessitent des partages d'informations entre les méthodes

- Ne faire qu'une seule classe *StepDef* contenant toutes les méthodes
- Utiliser des attributs de la classe pour modéliser l'état du système sous test
- Rapidement ingérable en raison de la taille de la classe
- Utiliser l'injection de dépendances de cucumber
- Utiliser « le World » : une classe injectée dans toutes les classes *StepDef*
- cucumber crée systématiquement une instance de World et les instances des classes *StepDef* à chaque scénario

Injection de dépendances

Dans le pom.xml

```
<dependency>  
  <groupId> io.cucumber </groupId>  
  <artifactId>cucumber-picocontainer</artifactId>  
  <version>${cucumber.version}</version>  
  <scope>test</scope>  
</dependency>
```

Injection de dépendances

Une classe pour « le World »

- Cette classe concentre les éléments à partager entre les pas des scénarios :
 - l'élément sous test
 - l'environnement de l'élément sous test
- et les méthodes pour accéder et stocker ces informations

```
public class MyWorld {  
  
    private Robot robot;  
  
    public MyWorld(){  
  
    public void setRobot(...)  
  
    public Robot getRobot()...  
}
```

Injection de dépendances

Dans les classes *StepDef*

- Chaque classe intègre un attribut pour mémoriser « le World »
- Le constructeur de la classe admet un World en argument

```
public class RobotFixture {  
  
    private MyWorld world;  
  
    public RobotFixture(MyWorld world) {  
        this.world = world;  
    }  
    ...  
}
```

Exercice

Spécifier un jeu de Morpion

- Dans un premier temps, spécifiez le fonctionnement du jeu du morpion au travers de Scénarios Gherkin
 - Utilisez pour ceci IntelliJ dans lequel vous aurez ajouté le plugin cucumber
 - Créez un projet maven MorpionAcceptanceTest
 - pas de répertoire src/main
 - répertoire src/test/java pour les classes de StepDef
 - répertoire src/test/resources pour les fichiers de Feature
 - intégrer les dépendances sur cucumber.io dans le pom.xml
 - Produisez les fichiers de Feature
 - détection de victoire
 - stratégie de placement de pion par l'IA
 - coup d'ouverture par l'IA
 - etc

Exercice

Tester une implémentation du jeu de Morpion

- Dans un second temps, récupérez un des projets morpion proposés sur moodle
 - Ouvrez ce projet dans une nouvelle fenêtre de IntelliJ
 - Effectuez un maven – install afin de déposer le jar du projet dans le repository maven local
- Dans une troisième étape, reprenez la fenêtre du projet MorpionAcceptanceTest
 - Ajoutez dans le pom.xml du projet MorpionAcceptanceTest, une dépendance sur le projet Morpion
 - Créez dans src/test/java les packages et classes permettant de d'interpréter les pas de scénario Gherkin sur l'application morpion
 - Exécutez les scénarios