

# De la spécification par l'exemple à la documentation vivante



*Isabelle BLASQUEZ*  
*@iblasquez*

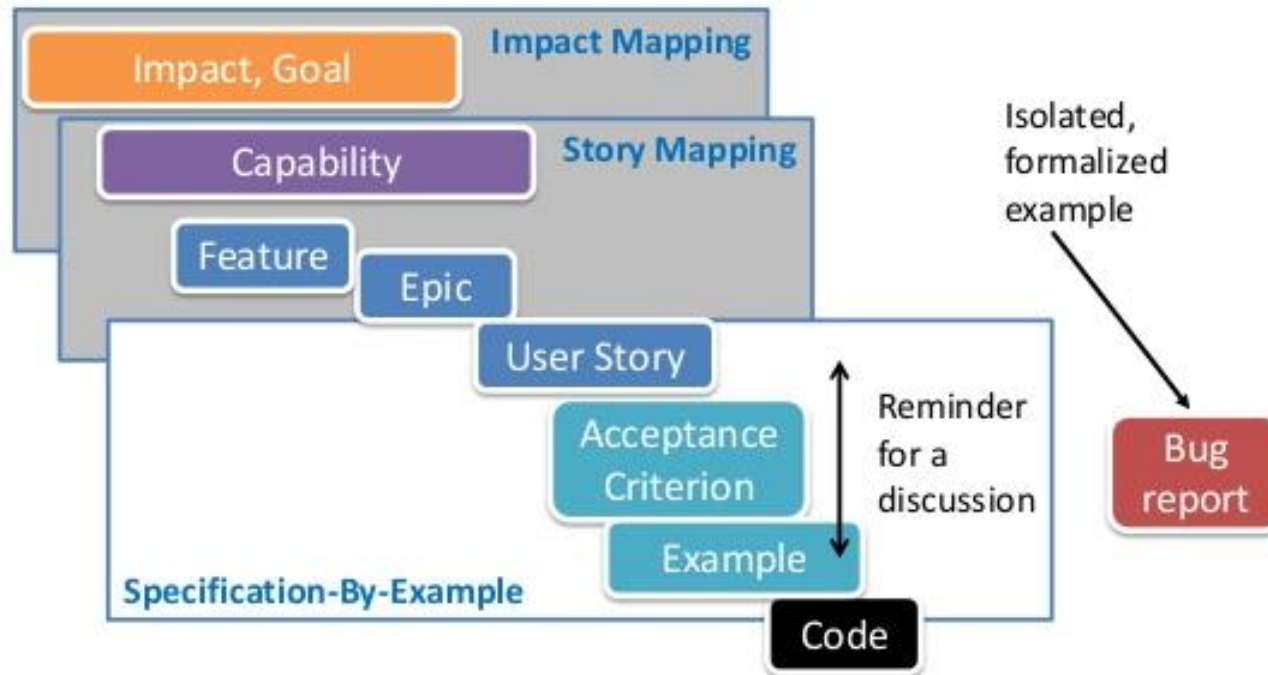
*2016*

# La story du POurQuoi au Comment

## Establishing a shared understanding

**Du POurQuoi ?**

Why?



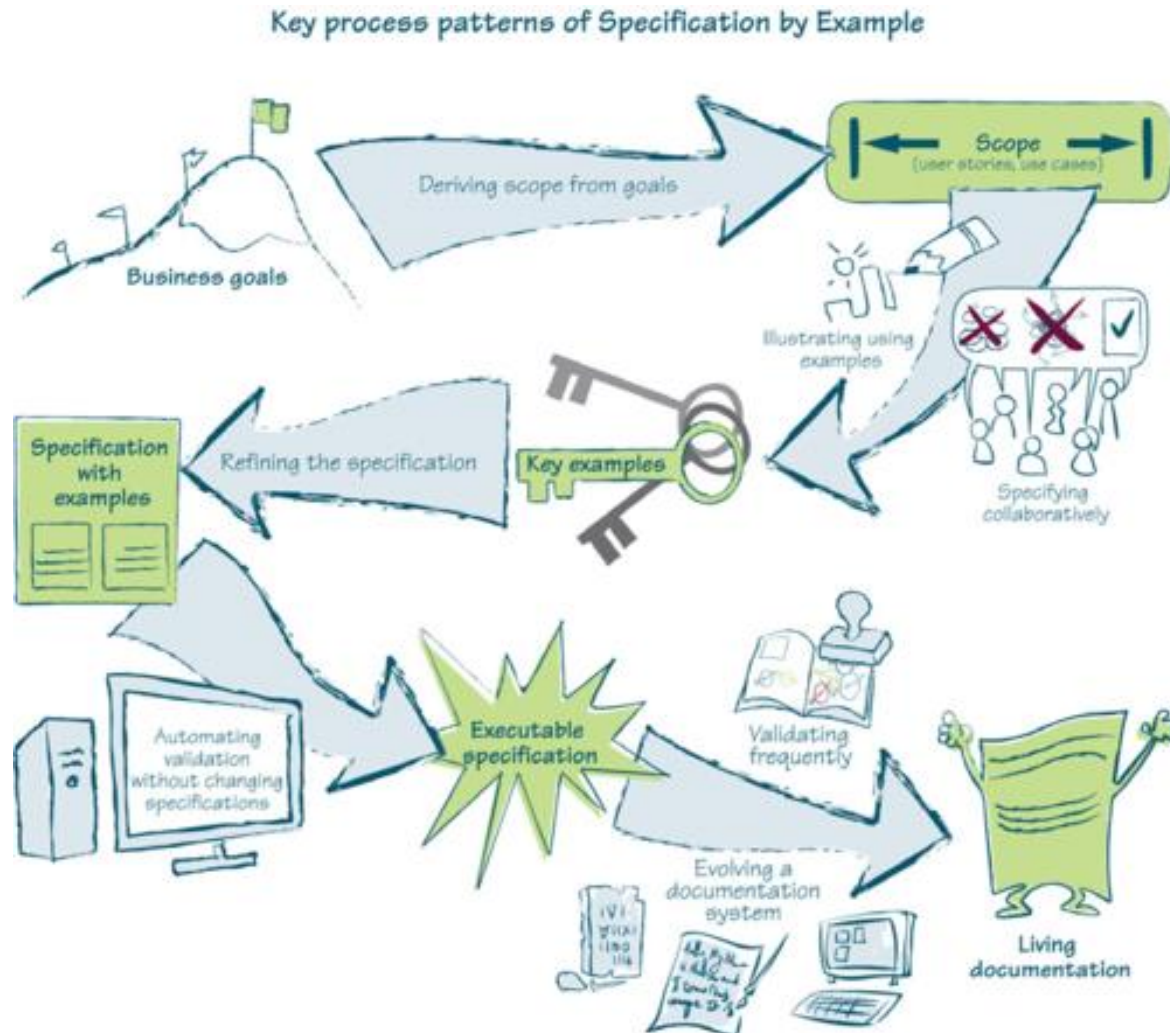
**Au Comment ?  
(Développeurs)**

Easier to define upfront

Harder to define upfront



# Une démarche *agile* de spécification par l'exemple ...



Cheminement de l'**objectif** (métier) à une **documentation vivante** au travers d'un ensemble de 7 patterns, qui permet de s'assurer que **le « bon » produit (right product)** sera effectivement livré.



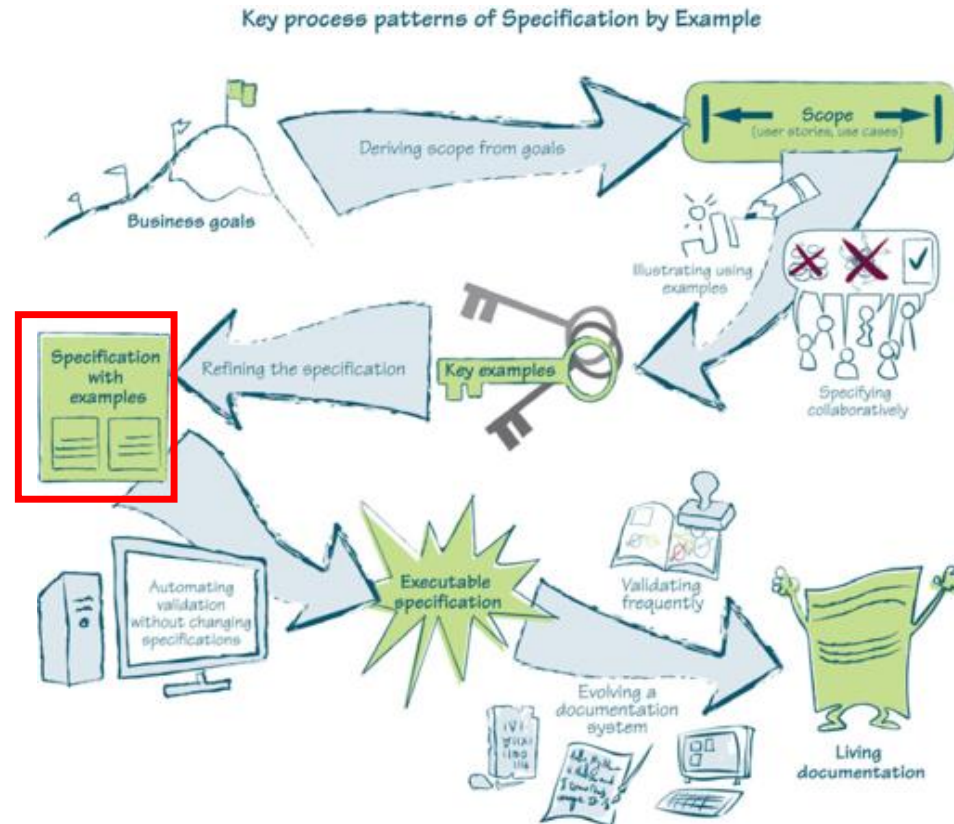
**Specification by example (SBE)** is a collaborative approach to defining **requirements** and **business-oriented functional tests** for software products based on capturing and illustrating requirements using **realistic examples** instead of abstract statements.

Extrait : [https://en.wikipedia.org/wiki/Specification\\_by\\_example](https://en.wikipedia.org/wiki/Specification_by_example)

# **Vers une spécification à base d'exemples (rappels)**

# Vers une spécification à base d'exemples ...

Avec de vraies valeurs, les scénarios abstraits décrivant le comportement de la condition d'acceptation deviennent des **exemples** concrets.



Ces **exemples**, décrits par des scénarios de tests, ne sont autre que de **tests d'acceptation**.



# En image : De la **condition** d'acceptation au **test** d'acceptation

On désire encourager les nouveaux clients à réaliser un achat en leur offrant 10% sur leur premier achat


```
public void TestInitialOrderDiscount()  
{  
    Customer newCustomer = new Customer();  
    Order newOrder = new Order(newCustomer);  
    newOrder.AddBook(  
        Catalog.Find("ISBN-0955683610")  
    );  
    Assert.Equals(33.75,  
        newOrder.Subtotal);  
}
```

Register as "bart\_bookworm"  
Go to "/catalog/search"  
Enter "ISBN-0955683610"  
Click "Search"  
Click "Add to Cart"  
Click "View Cart"  
Verify "Subtotal" is "\$33.75"

L'écriture des scénarios se fait en présentiel de **manière collaborative** (atelier des 3 Amigos)

L'expression des besoins se fait en « **langage naturel** » (*Ubiquitous Language*)

La grammaire **Etant donné/Quand/Alors** (*Gherkin*) peut permettre de structurer le scénario



**Etant donné** que le client n'a pas encore commandé

**Quand** le client ajoute dans un panier un livre avec un prix de 37,5 Euros

**Alors** le sous-total du panier est de 33,75 Euros

# Affiner les détails de la story en **structurant** son **comportement**

Dan North propose au travers du **BDD** (Behavior Driven Development) de **décrire le comportement d'une fonctionnalité** en le **structurant** avec une machine à états :

***Etat initial avant** exécution  
(précondition ou contexte)*

***Evénement** qui déclenche l'exécution*

***Etat après** l'exécution  
(postcondition ou résultat attendu)*

Il propose alors le **formalisme Given-When-Then**, appelé aussi « **gherkin** » pour exprimer ce comportement dans un **langage de spécification «naturel»** compréhensible de tous

**Etant donné** le contexte *et* la suite du contexte

**Quand** un événement survient

**Alors** on obtient un résultat *et* éventuellement un autre



On parle alors de **scénario** (de test).

# Détailier une condition d'acceptation par son *comportement* dans un « *langage naturel* »



**En tant que** maître de chien

**Je veux** pouvoir inscrire mon chien  
à une réunion de confirmation

**Afin** de soigner son pedigree

Condition  
d'acceptation  
au format BDD

## Inscription acceptée

**Etant donné** un maître de chien de race *et* un  
événement de confirmation prévu pour cette race

**Quand** le maître de chien inscrit un chien d'un âge  
autorisé à une confirmation

**Alors** l'inscription est acceptée *et* le maître de chien  
est informé de l'inscription *et* le nombre d'inscrits *est*  
incrémenté de 1

Contexte

Exécution  
de la story

Résultats  
attendus



# Passer au **test d'acceptation** en illustrant une condition d'acceptation par un exemple **concret**

Pattern AAA

**A**rrange

Inscription acceptée :

**Etant donné** Corinne propriétaire de Corsaire  
**et** une confirmation pour la race d'Épagneul annoncée pour le 15 Décembre avec 23 inscrits



**A**ct

**Quand** Corinne inscrit son épagneul Corsaire de 2 ans à la confirmation du 15 décembre

**A**ssert

**Alors** l'inscription de Corsaire est acceptée  
**et** le message « *Vous êtes bien inscrit à la confirmation Épagneul du 15 Décembre* » est envoyé à Corinne  
**et** le nombre d'inscrits passe à 24.



On parle alors de **test d'acceptation** (au format BDD)

On peut parfois être amené à écrire plusieurs tests d'acceptation (concret)  
pour illustrer une condition d'acceptation (abstrait)

# Autres formalismes possibles pour exprimer des exemples (scénarios de tests d'acceptation)

## Gherkin

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

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
```

## Tables

eg.Division

| numerator | denominator | quotient? |
|-----------|-------------|-----------|
| 10        | 2           | 5.0       |
| 12.6      | 3           | 4.2       |
| 22        | 7           | ~ = 3.14  |
| 9         | 3           | < 5       |
| 11        | 2           | 4 < _ < 6 |
| 100       | 4           | 33        |

```
|eg.Division|
|numerator|denominator|quotient?|
|10         |2           |5       |
|12.6       |3           |4.2     |
|100        |4           |33      |
```

## Mots Clés

```
*** Settings ***
Test Template      Calculate
Library            CalculatorLibrary

*** Test Cases ***
Additions          12 + 2 + 2      16
                  2 + -3           -1

Subtractions       12 - 2 - 2      8
                  2 - -3           5

Multiplication     12 * 2 * 2      48
                  2 * -3           -6

Division           12 / 2 / 2      3
                  2 / -3           -1

Calculation error  [Template]      Calculation should fail
                  kekkonen         Invalid button 'k'.
                  ${EMPTY}         Invalid expression.
                  1 / 0            Division by zero.

*** Keywords ***
Calculate
  [Arguments]      ${expression}    ${expected}
  Push buttons     C${expression}=
  Result should be ${expected}

Calculation should fail
  [Arguments]      ${expression}    ${expected}
  ${error} =       Should fail      C${expression}=
  Should be equal  ${expected}      ${error}
```

# **Cycle de Développement Agile :**

## **ATDD**

*(Acceptance Test  
Driven Development)*

# Acceptance Test Driven Development

Ateliers de  
**Spécification**  
(exemples)

**Discuss**



Tests  
Automatisés

**Distill**



**Develop**



**Demo**

**Right Product ?**

Validation du comportement attendu  
& Tests Exploratoires

**TDD or not ?**

 **FitNesse** Tables & Wiki

 **jbehave**

**cucumber**

 **specflow**  
Cucumber for .NET

**Scénarios**  
(Given -When-Then)

**Mots clés**

**ROBOT FRAMEWORK**

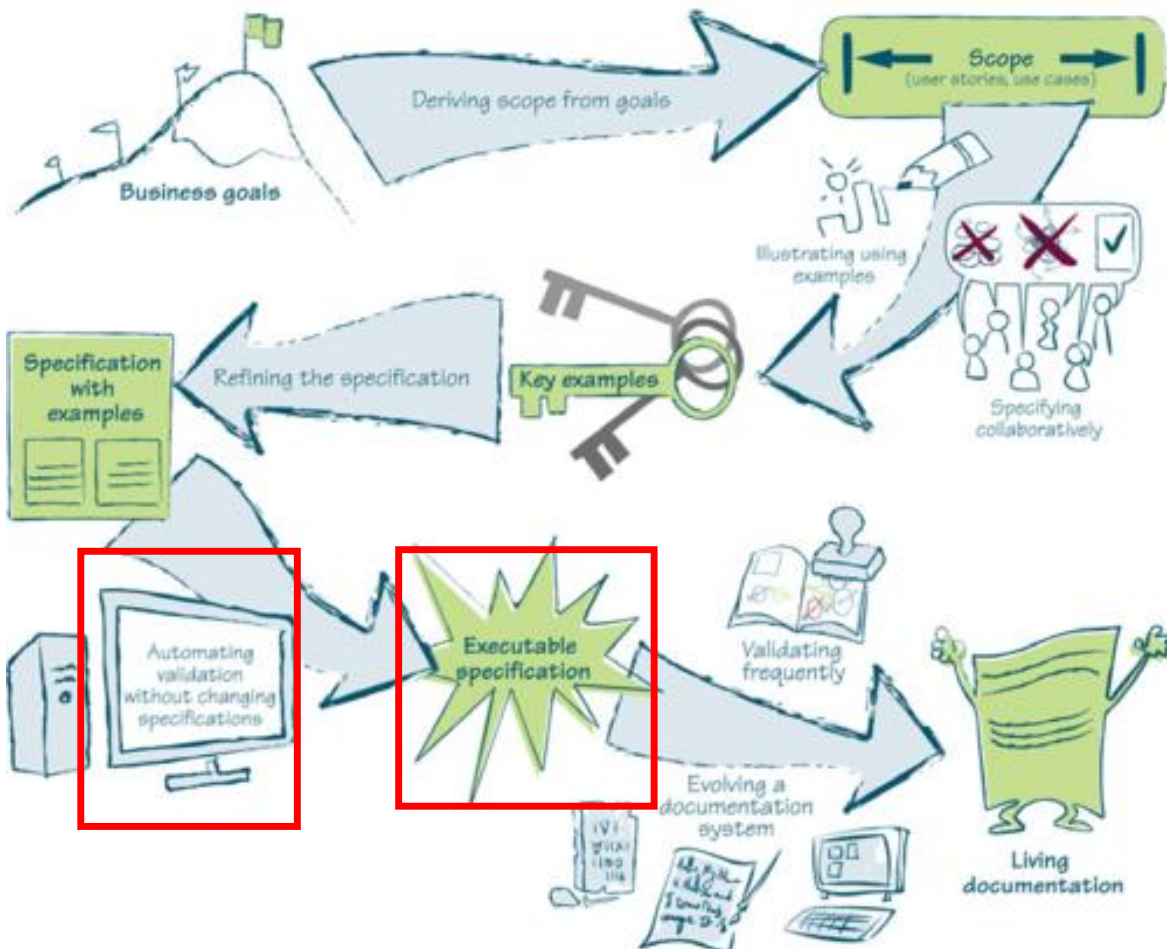
# **Zoom sur l'étape **Distill****

**(de la spécification par l'exemple à la spécification exécutable)**

# Avec de vraies valeurs, il est ensuite alors possible d'automatiser les exemples (tests) si on le souhaite afin d'obtenir une **spécification exécutable** !



Key process patterns of Specification by Example



**Automating validation  
without changing specification**

Une **Spécification avec des exemples** automatisés (tests) qui est compréhensible et accessible par tous les membres de l'équipe devient une **spécification exécutable**



# Panorama des différents outils pour l'automatisation des tests d'acceptation

## Tables & Wiki



| eg.Division |             |           |
|-------------|-------------|-----------|
| numerator   | denominator | quotient? |
| 10          | 2           | 5.0       |
| 12.6        | 3           | 4.2       |
| 22          | 7           | ~=3.14    |
| 9           | 3           | <5        |
| 11          | 2           | 4<_<6     |
| 100         | 4           | 33        |

The wiki markup for our table above is

```
|eg.Division|
|numerator|denominator|quotient?|
|10        |2          |5          |
|12.6      |3          |4.2        |
|100       |4          |33         |
```

Extrait : <http://www.fitnesse.org/>

## Mots Clés

([Data-driven test development](#))

## ROBOT FRAMEWORK

```
*** Settings ***
Test Template      Calculate
Library            CalculatorLibrary

*** Test Cases ***
Additions          12 + 2 + 2    16
                  2 + -3         -1

Subtractions       12 - 2 - 2    8
                  2 - -3         5

Multiplication     12 * 2 * 2    48
                  2 * -3         -6

Division           12 / 2 / 2    3
                  2 / -3         -1

Calculation error  [Template]    Calculation should fail
                  kekkonen       Invalid button 'k'.
                  ${EMPTY}       Invalid expression.
                  1 / 0          Division by zero.

*** Keywords ***
Calculate
[Arguments]        ${expression}  ${expected}
Push buttons       C${expression}=
Result should be   ${expected}

Calculation should fail
[Arguments]        ${expression}  ${expected}
${error} =         Should fail    C${expression}=
Should be equal    ${expected}    ${error}
```

Extrait : <http://robotframework.org/>

## Scénarios (BDD) (Given-When-Then)



### 1. Describe behaviour in plain text

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

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
```

Extrait : <https://cucumber.io/>

# Les différentes étapes de la spécification par l'exemple à la spécification exécutable avec

## 1. Write story

Plain text

Scenario: A trader is alerted of status

Given a stock and a threshold of 15.0

When stock is traded at 5.0

Then the alert status should be OFF

When stock is traded at 16.0

Then the alert status should be ON

## 2. Map steps to Java

POJO

```
public class TraderSteps {  
    private TradingService service; // Injected  
    private Stock stock; // Created  
  
    @Given("a stock and a threshold of $threshold")  
    public void aStock(double threshold) {  
        stock = service.newStock("STK", threshold);  
    }  
    @When("the stock is traded at price $price")  
    public void theStockIsTraded(double price) {  
        stock.tradeAt(price);  
    }  
    @Then("the alert status is $status")  
    public void theAlertStatusIs(String status) {  
        assertThat(stock.getStatus().name(), equalTo(status));  
    }  
}
```

**Automatisation :**  
steps du scénario  
transformées en code  
pour pouvoir être exécutées

Assertion ⇒  
*c'est bien un test d'acceptation !*

## Configuration

Indispensable pour mettre  
en place l'automatisation

## 3. Configure Stories

Only once

```
public class TraderStories extends JUnitStories {  
    public Configuration configuration() {  
        return new MostUsefulConfiguration()  
            .useStoryLoader(new LoadFromClasspath(this.getClass()))  
            .useStoryReporterBuilder(new StoryReporterBuilder()  
                .withCodeLocationFromClass(this.getClass())  
                .withFormats(CONSOLE, TXT, HTML, XML));  
    }  
  
    public List<CandidateSteps> candidateSteps() {  
        return new InstanceStepsFactory(configuration(),  
            new TraderSteps(new TradingService()))  
            .createCandidateSteps();  
    }  
  
    protected List<String> storyPaths() {  
        return new StoryFinder().findPathsFromClass(this.getClass(),  
            "**/*.*story");  
    }  
}
```



**Ecriture  
du code métier  
de production  
permettant  
d'implémenter  
le périmètre  
fonctionnel du test**

## 4. Run Stories

With any of



**Exécution de l'automatisation :  
lancement du test**

## 5. View Reports

HTML

Scenario: A trader is alerted of status

Given a stock and a threshold of 15.0

When stock is traded at 5.0

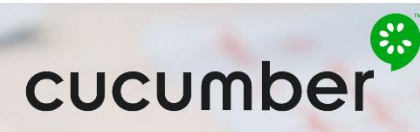
Then the alert status is OFF

When stock is traded at 16.0

Then the alert status is ON

**Spécification  
exécutable**

# De la spécification par l'exemple à la spécification exécutable avec (même principe que précédemment)



**Cucumber** *behaviour driven development  
with elegance and joy*

## 1: Describe behaviour in plain text

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

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
```

## 2: Write a step definition in Ruby

```
Given /I have entered (.*) into the calculator/ do |n|
  calculator = Calculator.new
  calculator.push(n.to_i)
end
```

## 3: Run and watch it fail

```
$ cucumber features/addition.feature
Feature: Addition # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
Scenario: Add two numbers # features/addition.feature
  Given I have entered 50 into the calculator # features/step_definitions/calculator_steps.rb:2:in `Given I have entered 50 into the calculator'
    uninitialized constant Calculator (NameError)
    ./features/step_definitions/calculator_steps.rb:2:in `Given I have entered 50 into the calculator'
    features/addition.feature:7:in `Given I have entered 50 into the calculator'
  And I have entered 70 into the calculator # features/step_definitions/calculator_steps.rb:2:in `Given I have entered 70 into the calculator'
  When I press add # features/step_definitions/calculator_steps.rb:2:in `When I press add'
  Then the result should be 120 on the screen # features/step_definitions/calculator_steps.rb:2:in `Then the result should be 120 on the screen'
```

## 4. Write code to make the step pass

```
class Calculator
  def push(n)
    @args ||= []
    @args << n
  end
end
```

## 5. Run again and see the step pass

```
$ cucumber features/addition.feature
Feature: Addition # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
Scenario: Add two numbers # features/addition.feature
  Given I have entered 50 into the calculator # features/step_definitions/calculator_steps.rb:2:in `Given I have entered 50 into the calculator'
  And I have entered 70 into the calculator # features/step_definitions/calculator_steps.rb:2:in `And I have entered 70 into the calculator'
  When I press add # features/step_definitions/calculator_steps.rb:2:in `When I press add'
  Then the result should be 120 on the screen # features/step_definitions/calculator_steps.rb:2:in `Then the result should be 120 on the screen'
```

## 6. Repeat 2-5 until green like a cucumber

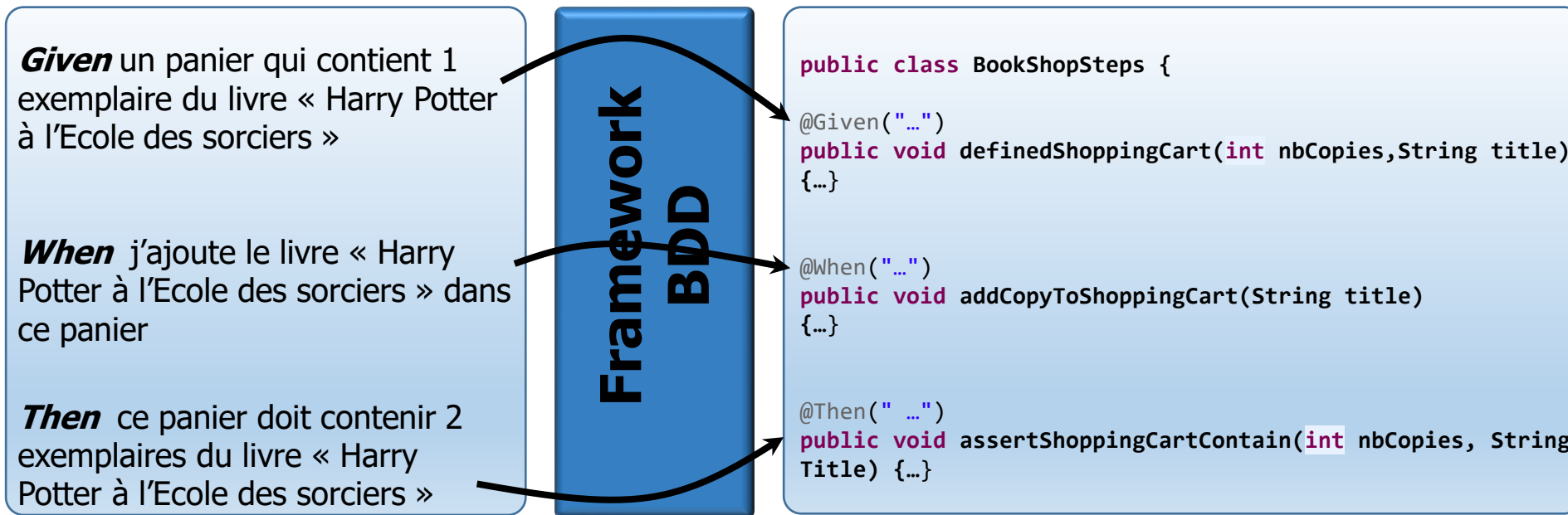
```
$ cucumber features/addition.feature
Feature: Addition # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
Scenario: Add two numbers # features/addition.feature
  Given I have entered 50 into the calculator # features/step_definitions/calculator_steps.rb:2:in `Given I have entered 50 into the calculator'
  And I have entered 70 into the calculator # features/step_definitions/calculator_steps.rb:2:in `And I have entered 70 into the calculator'
  When I press add # features/step_definitions/calculator_steps.rb:2:in `When I press add'
  Then the result should be 120 on the screen # features/step_definitions/calculator_steps.rb:2:in `Then the result should be 120 on the screen'
```



# Principe de fonctionnement des frameworks BDD pour l'automatisation des tests d'acceptation



... et bien d'autres ...



- ✓ Les outils de BDD permettent de traduire un scénario en langage naturel en appels de méthodes.
- ✓ La grammaire **Given/When/Then** (appelé langage **Gherkin**) permet de réaliser le mapping entre les « étapes » du scénario et les « steps » du code.

# Un exemple de spécification exécutable obtenue avec FitNesse

Au delà d'une simple démarche d'automatisations des tests, il faut percevoir les spécifications exécutables comme une véritable opportunité de rapprocher les populations techniques et fonctionnelles autour d'une vision partagée et non ambiguë du produit logiciel.

## Final expression of our example

|         |      |
|---------|------|
| do with | bank |
|---------|------|

Our first business rule says that a new account should have a balance of 0.00 dollars.

|       |                         |             |             |                   |           |   |             |
|-------|-------------------------|-------------|-------------|-------------------|-----------|---|-------------|
| open  | checking                | account     | 12345-67890 | under the name of | Spongebob | ? | Squarepants |
| check | that balance of account | 12345-67890 | is          | \$0.00            |           |   |             |

Our next rule says that the bank should not take any fees when we deposit money in our account.

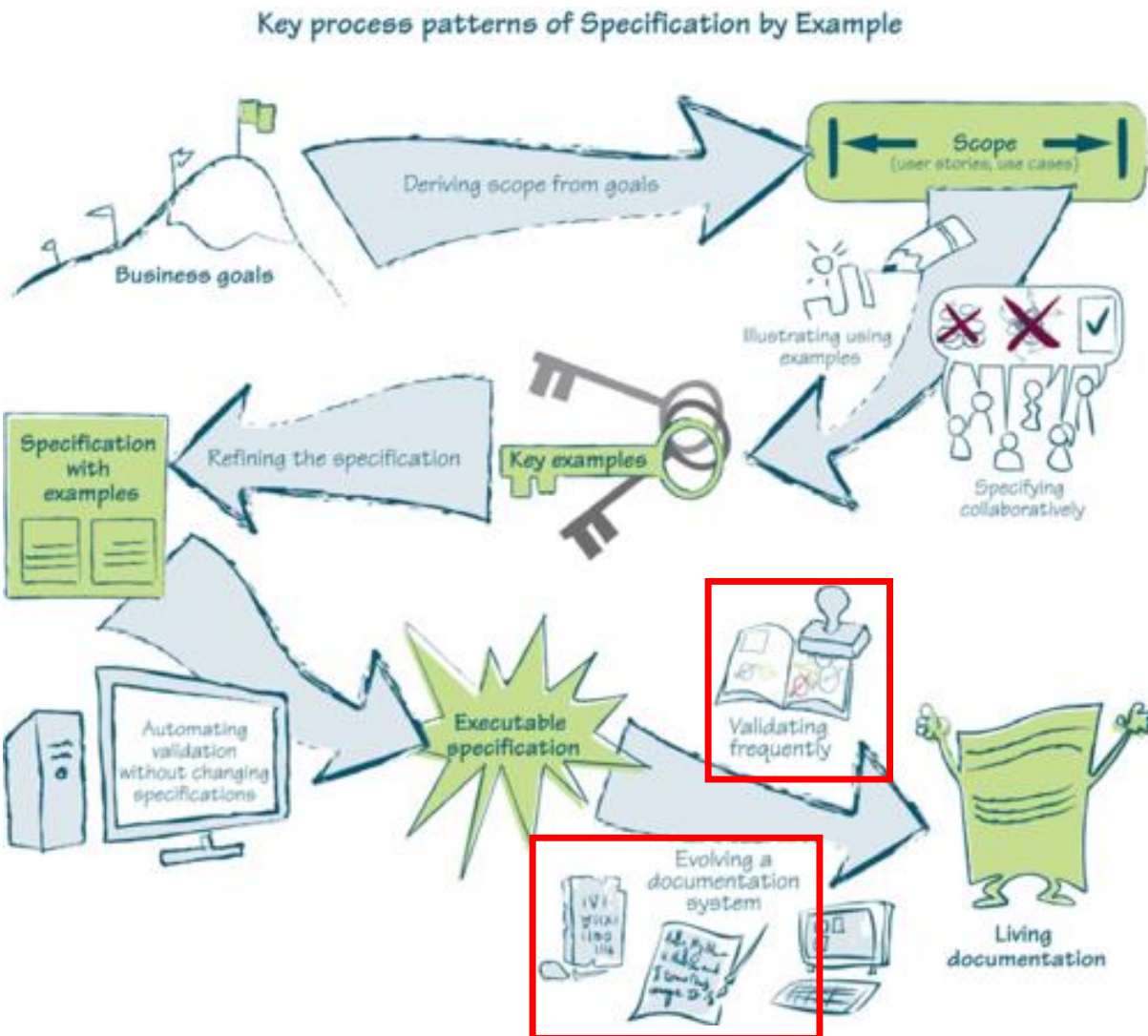
|         |                         |             |             |          |  |
|---------|-------------------------|-------------|-------------|----------|--|
| deposit | \$100.00                | in account  | 12345-67890 |          |  |
| check   | that balance of account | 12345-67890 | is          | \$100.00 |  |

The following rule says that a customer should be able to withdraw funds if the balance of his account is sufficient.

|          |                         |              |              |             |  |
|----------|-------------------------|--------------|--------------|-------------|--|
| withdraw | \$50.00                 | from account | 12345-67890  |             |  |
| check    | that balance of account | 12345-67890  | is           | \$50.00     |  |
| reject   | withdraw                | \$75.00      | from account | 12345-67890 |  |
| check    | that balance of account | 12345-67890  | is           | \$50.00     |  |
| accept   | withdraw                | \$25.00      | from account | 12345-67890 |  |

*Un exemple de spécifications exécutables venant d'être exécutées et en succès.*

# Et pour finir le cheminement de la Spécification par l'exemple



**Validating frequently**

**Evolving a documentation system**

Proposer une **documentation vivante** :  
facile à comprendre, cohérente et organisée



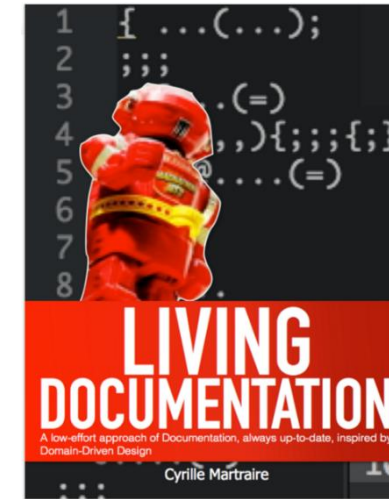
# En savoir un peu plus sur la documentation vivante...



Living Documentation : vous allez aimer la documentation ! (Cyrille Martraire)

*A visualiser sur :*

<https://www.youtube.com/watch?v=Tw-wcps7WqU>



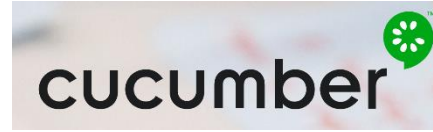
*Livre disponible sur :*

<https://leanpub.com/livingdocumentation>



CommitStrip.com

# Prise en main de Cucumber ...



## 🔗 Tutoriel de prise en main de Cucumber

Ce tutoriel s'inspire de l'exemple Shakespeare du *Kata Cucumber/Mockito* de Sébastien Mosser

**Cucumber** est un framework de tests pour le Behavior Driven Development, initialement développé en Ruby, mais proposant aujourd'hui des implémentations possibles pour de nombreux autres langages de programmation. Le site de référence est : [cucumber.io](http://cucumber.io).

Dans l'écosystème Java, **Cucumber** est aujourd'hui un des frameworks BDD les plus utilisés.

Comme tout framework adapté au BDD, **Cucumber** permet de transformer les scénarii d'une story (écrits sous forme d'exemples en *langage naturel* au format *Gherkin*) en tests java automatisés. Dans le principe, cette transformation est possible à l'aide du framework de tests **JUnit**. Chaque **étape d'un scénario** est implémentée comme une **méthode java**, appelée **step**. Le lien entre la description textuelle de l'étape et le code Java de la step est réalisé via des annotations.

Dans ce tutoriel, nous verrons comment :

- Installer le plug-in Cucumber-Eclipse
- Mettre en place votre premier projet Cucumber
  - 1. Créer un projet Maven
  - 2. Configurer le `pom.xml` pour Cucumber
  - 3. Décrire le comportement en langage naturel ( `.feature` )
  - 4. Configurer le lanceur de tests
  - 5. Implémenter le code de test des *steps* (méthodes java)
  - 6. Implémenter le code métier de l'application

Mais aussi, comment :

- Paramétrer les steps à l'aide d'expressions régulières
- Alléger la lecture d'un scénarios ( `And` et `But` )
- Paramétrer un scénario
- Factoriser des scénarios
- Paramétrer le lanceur de test à partir de `@CucumberOptions`
- Produire un *beau* rapport à l'aide du plug-in `cucumber-reporting`
- Remarques complémentaires

Tutoriel disponible sur : [https://github.com/iblasquez/tuto\\_bdd\\_cucumber](https://github.com/iblasquez/tuto_bdd_cucumber)

# Et dans la vraie vie ?

## Projects Using Cucumber

Marcello Nuccio edited this page on 22 Apr 2015 · 3 revisions

Liste de projets utilisant Cucumber accessibles depuis :  
<https://github.com/cucumber/cucumber/wiki/Projects-Using-Cucumber>

Here is a list of projects using Cucumber. I find reading them to be a great learning experience.

In alphabetical order:

- [BehatMage](#)
- [Broth](#)
- [bsmi](#)
- [CarrierWave](#)
- [Chef](#)
- [Chits](#)
- [Courgette](#)
- [CF FCQ](#)
- [Diaspora](#)
- [drush-make-ci](#)
- [Folioapp](#)
- [GitLab](#)
- [Jekyll](#)
- [Jeweler](#)
- [Libra Open Access](#)
- [OERPScenario](#) (linked to OpenERP)
- [One Click Orgs](#)
- [PHP Sasl](#)
- [RadiantCMS](#)
- [Rails directory](#)
- [Redcar](#) (see `/plugins/*/features`)
- [rigse](#)
- [rps-challenge](#)
- [TimeFliesBy](#)
- [Vdebug](#)
- [WebJam](#)
- [WontoMedia](#)

Jetez par exemple un petit coup d’œil aux features de Gitlab consultables sur  
<https://github.com/gitlabhq/gitlabhq/tree/master/features>

Branch: master ▾


gitlabhq / features /

Create new file

Upload files


Find file

History

 Jacob Schatz Merge branch '20840-getting-started-better-empty-state-for-issues-vie... ...


Latest commit **a7616e0** a day ago

..

 admin


Revert "Revert "Merge branch 'issue\_3946' into 'master' ""

5 months ago

 dashboard


Fix spinach tests

22 days ago

 explore


Fix trending projects Spinach failure

a month ago

 group


Updates from last code review.

9 months ago

 profile


implements reset incoming email token on issues modal and account page,

16 days ago

 project


Merge branch 'fix-404-on-network-when-entering-a-nonexistent-git-revi...

15 days ago

 snippets


Move 'Explore Snippets' Spinach feature to Rspec

15 days ago

 steps


Merge branch '20840-getting-started-better-empty-state-for-issues-vie...

a day ago

 support


Disable warming of the asset cache in Spinach tests under CI

a month ago

 abuse\_report.feature


Streamline the "Report button"

a year ago

 groups.feature


Move edit group scenario to rspec and refactor groups\_spec

a month ago

 invites.feature


Add spinach tests around accepting and declining invitations.

2 years ago

 search.feature

Fixed logout tests

5 months ago

 user.feature

Fix specs

9 months ago



# Poursuivre votre réflexion sur le BDD

