Le développement piloté par le comportement

Behavior Driven Development (BDD)

Alexane / Armand / Claire / Embarek / Franck / Jackson / Noémie

G6S2 - 26/11/15

Définition

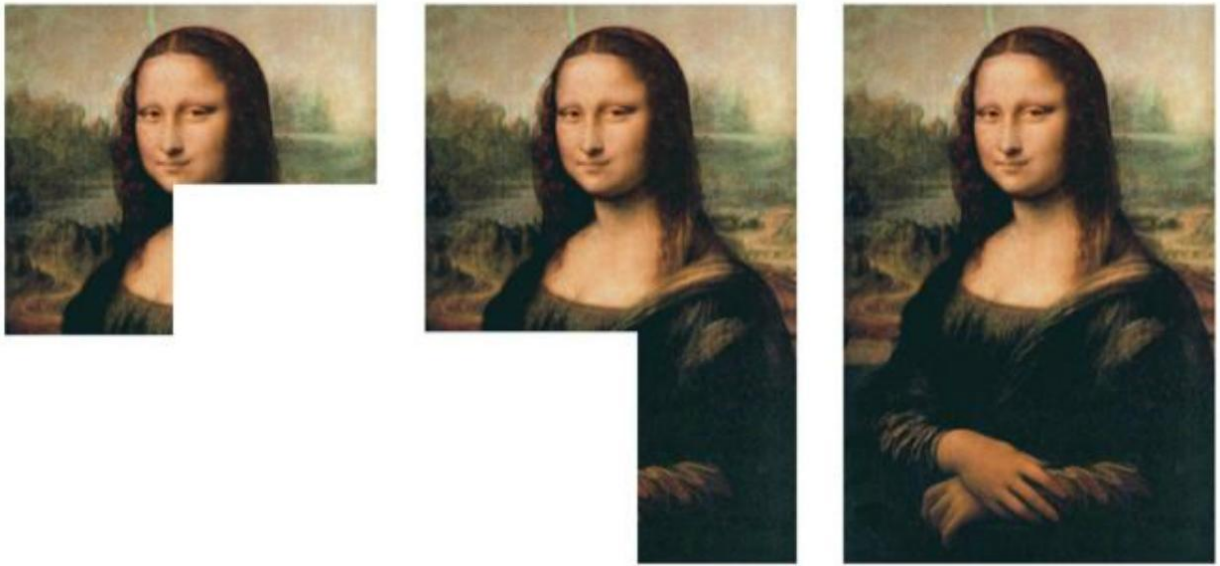
Méthode agile qui permet de faciliter la communication entre le client et le développeur par l'utilisation d'un langage simple où le besoin est exprimé sous forme de fonctionnalités

---

"Je veux un tableau où l'utilisateur peut définir sa taille pour mon puissance 4"

Faire une saisie pour l'utilisateur et créer le tableau

Méthode classique, cycle en V



Changement fonctionnel difficile en cours

Méthode agile, développement par itérations

Changement fonctionnel facile, même en cours de route

Caractéristiques d’une fonctionnalité …

- Un contexte
- Un service rendu
- Un bénéfice métier

… qui comporte un ou plusieurs scénarios de tests

- Quel est le contexte ?
- Quels sont les évènements ?
- Quel est le résultat attendu ?

Exemples de fonctionnalités

Fonctionnalité 1 - saisir la taille du tableau

En tant qu'utilisateur
Je veux pouvoir saisir la taille du tableau
De telle sorte que je puisse personnaliser la grille

Fonctionnalité 2 - acheter des concombres

En tant que client du magasin
Je veux pouvoir acheter des concombres
De telle sorte que je puisse préparer une salade

Exemples de scénarios pour la fonctionnalité 1

Scénario 1.1 - saisir une taille positive

Étant donné que je souhaite créer un tableau
Quand je saisis une taille positive
Alors le tableau est créé

Scénario 1.2 - saisir une taille négative

Étant donné que je souhaite créer un tableau
Quand je saisis une taille négative
Alors on m'informe que la taille est incorrecte

Exemples de scénarios pour la fonctionnalité 2

Scénario 2.1 - acheter un concombre bleu
    Étant donné que j'aime le bleu
    Quand je demande un concombre bleu
    Alors je suis informé que ça n'existe pas

Scénario 2.2 - acheter un concombre vert
    Étant donné qu'un concombre vert est disponible
    Quand j'achète un concombre vert
    Alors on me donne un concombre vert

Avantages

- Langage accessible à tous sans ambiguïté
- Suppression des bugs lors de la conception
- Moins de gaspillage ressources

Inconvénients

- Participation du client obligatoire
- Laborieux, très manuel et beaucoup de compilations

Démonstration sous Cucumber

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/additi
    Given I have entered 50 into the calculator      # features/step_d
      uninitialized constant Calculator (NameError)
      ./features/step_definitons/calculator_steps.rb:2:in `Given /
      features/addition.feature:7:in `Given I have entered 50 into
    And I have entered 70 into the calculator        # features/step_d
    When I press add                                 # features/additi
    Then the result should be 120 on the screen      # features/additi
```

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/additi
    Given I have entered 50 into the calculator      # features/step_d
    And I have entered 70 into the calculator        # features/step_d
    When I press add                                 # features/additi
    Then the result should be 120 on the screen      # features/additi
```

6. Repeat 2-5 until green like a cuke

```
$ 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/additi
    Given I have entered 50 into the calculator      # features/step_d
    And I have entered 70 into the calculator        # features/step_d
    When I press add                                 # features/step_d
    Then the result should be 120 on the screen      # features/step_d
```

Démonstration sous Cucumber

Conclusion

- Le BDD est une méthode de travail centrée sur le besoin du client
- Elle est une alternative plus humaine aux méthodes classiques
- Toutefois, elle n'est efficace que si le client s'implique réellement dans la conception du logiciel

Références

[Jean-François Lépine]

[Connect]

[Cucumber]

Merci pour votre attention !