Tests d'acceptation et BDD (*Behavior Driven Development*)

Guy Tremblay
Professeur

Département d'informatique
UQAM
http://www.labunix.uqam.ca/~tremblay

13–20 octobre 2016

•

Contenu

1 Introduction : Tests unitaires de style TDD vs. BDD

2 Que sont les tests d'acceptation ?

3 Les tests d'acceptation et l'approche BDD

4 Des outils pour le BDD

5 D'autres outils pour les tests d'acceptation

6 Un exemple plus détaillé avec `cucumber:biblio`

7 Conclusion

•

1. Introduction : Tests unitaires de style TDD vs. BDD

Suggestion de D. North (2003)

Le nom d'une méthode de test devrait être une phrase qui nous éclaire sur le comportement de la méthode testée, y compris en cas de traitement d'erreur ou d'exception

Le nom d'une méthode de test devrait être une phrase qui nous éclaire sur le comportement de la méthode testée, y compris en cas de traitement d'erreur ou d'exception

- Pour bien nommer les cas de test, il faut tenir compte que pour une méthode du code à tester, par exemple, `retirer`, on aura plusieurs cas de tests à écrire — plusieurs méthodes de tests — et ce pour tester différents aspects, différentes situations : retirer une partie, retirer tout, retirer plus que le solde, etc. On ne pourra pas simplement appeler ces méthodes `tester_retirer`, car ce ne serait pas assez spécifique.
- De plus, il faut aussi avoir des cas de test les plus indépendants possibles, pour bien cerner/identifier les erreurs problèmes possibles quand quelque chose ne fonctionne pas.
- Mais ceci a amené une autre question, à savoir : quel est un nom approprié, significatif, pour un cas de test (une méthode de test) ? Parce que les règles pour le nommage des méthodes habituelles ne s'appliquent pas, style, prédicat pour les observateur, verbe pour les actions ! Le nom de la méthode de test doit plutôt nous renseigner sur l'aspect du module/classe/méthode qui est testé.

Le nom d'une méthode de test devrait être une phrase qui décrit le comportement attendu : Contre-exemple

JUnit 3.0

```
public void testRetirer() {
  c.retirer( c.solde() );

  assertEquals( 0, c.solde() );
}
```


Contre-exemple
JUnit 3.0
public void testRetirer() {
c.retirer(c.solde());
assertEquals(0, c.solde());
}

-

Le nom d'une méthode de test devrait être une phrase qui décrit le comportement attendu : Contre-exemple

JUnit 4.0

```
@Test
public void retirer() {
  c.retirer( c.solde() );

  assertEquals( 0, c.solde() );
}
```


Contre-exemple
JUnit 4.0
@Test
public void retirer() {
c.retirer(c.solde());
assertEquals(0, c.solde());
}

-

Le nom d'une méthode de test devrait être une phrase qui décrit le comportement attendu : Suggestion

Convention suggérée pour les noms des méthodes de test :

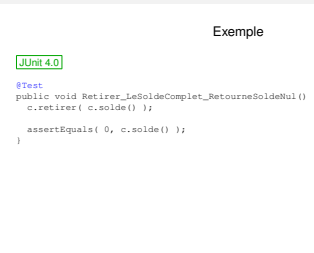
`NomDeMéthode_ÉtatTesté_RésultatAttendu`

- Motivations de cette convention :
  - Le nom du test devrait exprimer une exigence spécifique
  - Le nom du test devrait inclure les données ou l'état, tant en entrée qu'en sortie
  - Le nom du test devrait inclure le nom de la méthode ou classe testée
- Autre motivation :
  - Le nom du test devrait nous aider à comprendre le comportement attendu y compris (surtout !) lorsque le test «échoue».

Le nom d'une méthode de test devrait être une phrase qui décrit le comportement attendu : Exemple

JUnit 4.0

```
@Test
public void Retirer_LeSoldeComplet_RetourneSoldeNul() {
  c.retirer( c.solde() );

  assertEquals( 0, c.solde() );
}
```

- Donc, en Java avec `JUnit`, ça peut donner un nom de méthode qui aurait l'allure suivante, donc pas nécessairement facile à lire, et encore moins à écrire ☹

Le nom d'une méthode de test devrait être une phrase qui décrit le comportement attendu : Exemple

Ruby (`Test::Unit`)

```ruby
def test_retirer_le_solde_complet_retourne_solde_nul
  @c.retirer( @c.solde )

  assert_equal 0, @c.solde
end
```


- En Ruby, avec `Test::Unit`, ça donnerait cela — sauf pour les noms de classe, on utilise pas le *CamelCase* en Ruby. Là non plus, ce n'est pas l'idéal.

Le nom d'une méthode de test devrait être une phrase qui décrit le comportement attendu : Exemple

Ruby (à la `RSpec`)

```
test "retirer le solde complet retourne solde nul" do
  @c.retirer( @c.solde )

  assert_equal 0, @c.solde
end
```


Exemple
Ruby (à la RSpec)
test "retirer le solde complet retourne solde nul" do
@c.retirer(@c.solde)
assert_equal 0, @c.solde
end

- Par contre, en Ruby, il y a plein de «trucs» qu'on peut faire, difficile ou impossible à faire en Java.
- Par exemple, cette façon d'écrire le nom du test ne serait-elle pas plus simple à écrire et à lire ?

Suggestion de D. North (2003)

Débuter le nom d'une méthode de test par `should` aide à ce que le test soit mieux ciblé

Exemple :

Retirer

- devrait retourner un solde nul lorsqu'on retire tout
- devrait échouer lorsqu'on retire plus que le solde courant
- ...

Et aussi : quand le test échoue, un nom de test expressif aide à mieux comprendre où est le problème

Débuter le nom d'une méthode de test par `should` aide à ce que le test soit mieux ciblé
Exemple :
Retirer
- devrait retourner un solde nul lorsqu'on retire tout
- devrait échouer lorsqu'on retire plus que le solde courant
- ...
Et aussi : quand le test échoue, un nom de test expressif aide à mieux comprendre où est le problème

- ...

JBehave (North, 2003)

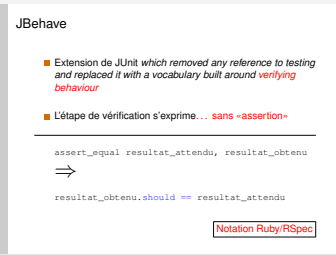
- Extension de JUnit *which removed any reference to testing and replaced it with a vocabulary built around verifying behaviour*
- L'étape de vérification s'exprime... sans «assertion»

```
assert_equal resultat_attendu, resultat_obtenu
```

$\Rightarrow$

```
resultat_obtenu.should == resultat_attendu
```

Notation Ruby/RSpec

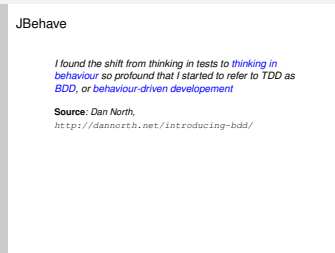


• Donc, North, en 2003, a développé JBehave, un outil permettant de spécifier des cas de tests en Java en s'inspirant de ces constatations et suggestions.
• Dans ce qui suit, je vais toutefois l'illustrer en Ruby, avec `RSpec`, avec lequel je suis plus familier... et qui a eu plus de succès que JBehave.

JBehave (North, 2003)

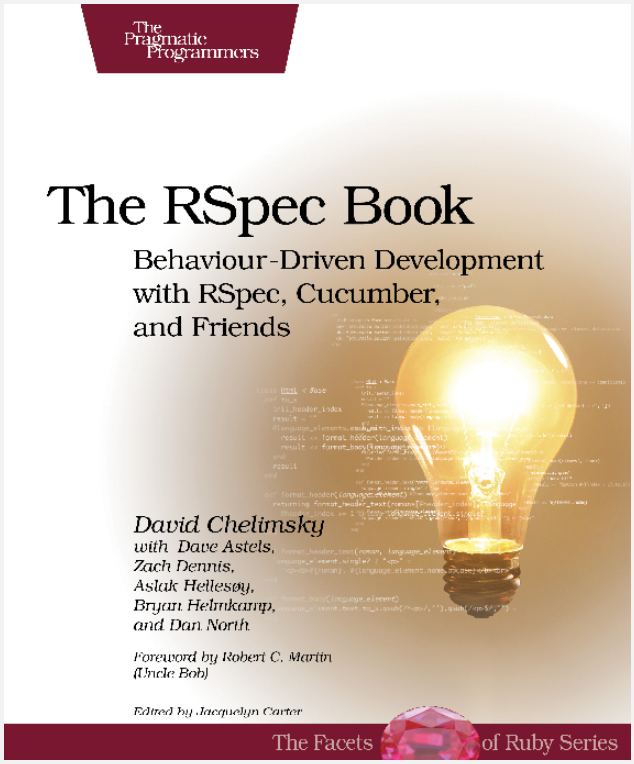
I found the shift from thinking in tests to thinking in behaviour so profound that I started to refer to TDD as BDD, or behaviour-driven developement

Source*: Dan North,*
http://dannorth.net/introducing-bdd/

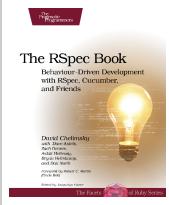


-

RSpec (Chelimsky, North et al., 2007)



Cadre de tests Ruby inspiré de JBehave



Cadre de tests Ruby inspiré de JBehave

-

Exemple RSpec
Définition de `Compte`

```ruby
class Compte
  attr_reader :solde, :client

  def initialize( client, solde_init )
    @client, @solde = client, solde_init
  end

  def deposer( montant )
    @solde += montant
  end

  def retirer( montant )
    fail "Solde insuffisant" if montant > solde

    @solde -= montant
  end
end
```


Définition

```ruby
class Compte
  attr_reader :solde, :client

  def initialize( client, solde_init )
    @client, @solde = client, solde_init
  end

  def deposer( montant )
    @solde += montant
  end

  def retirer( montant )
    fail "Solde insuffisant" if montant > solde

    @solde -= montant
  end
end
```

- Donc, soit à nouveau notre classe pour un `Compte` bancaire simple.

Exemple RSpec

Spécification de `Compte` (1)... avec «devrait»

```
describe Compte do
  before(:each) { @c = Compte.new( "Guy T.", 100 ) }

  describe ".new" do
    it "devrait creer un compte avec le solde initial indique" do
      @c.solde. should == 100
    end
  end

  describe "#deposer" do
    it "devrait ajouter le montant indique au solde du compte" do
      solde_initial = @c.solde

      @c.deposer( 100 )
      @c.solde. should == solde_initial + 100
    end
  end
```


- Voici des tests équivalents à ceux vus précédemment, avec en plus des tests pour `retirer`.
- Éléments de «convention» pour décrire les tests :
  - Le premier niveau de `describe` indique pour quelle classe on est en train de spécifier le comportement.
  - Chaque niveau interne de `describe` indique alors la méthode décrite/spécifiée — avec le préfixe «`.`» pour les méthodes de classe et avec le préfixe «`#`» pour les méthodes d'instance.
  - Un `it` représente un cas de test.
- Les assertions utilisées sont exprimées d'une façon différente. Dans les premières versions de RSpec, on utilisait des `should` :

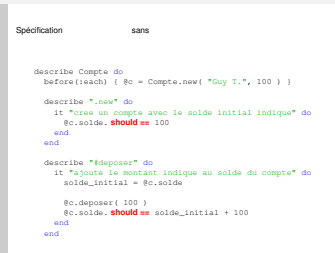
```
assert_equal solde_initial+100, @c.solde
```

```
@c.solde.should == solde_initial + 100
```

Exemple RSpec

Spécification de `Compte` (1)... sans «devrait»

```
describe Compte do
  before(:each) { @c = Compte.new( "Guy T.", 100 ) }

  describe ".new" do
    it "cree un compte avec le solde initial indique" do
      @c.solde. should == 100
    end
  end

  describe "#deposer" do
    it "ajoute le montant indique au solde du compte" do
      solde_initial = @c.solde

      @c.deposer( 100 )
      @c.solde. should == solde_initial + 100
    end
  end
```

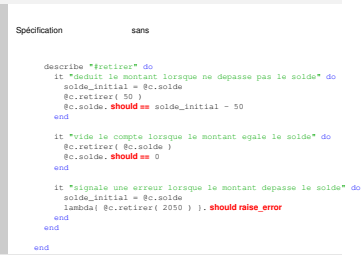



- Certains suggéraient, les premiers temps, que les tests débutent avec «should». Mais maintenant, c'est considéré comme une mauvaise pratique : pourquoi dire «devrait faire X» quand on peut dire tout aussi clairement et simplement «fait X».

Exemple RSpec

Spécification de `Compte` (2)... sans «devrait»

```
  describe "#retirer" do
    it "deduit le montant lorsque ne depasse pas le solde" do
      solde_initial = @c.solde
      @c.retirer( 50 )
      @c.solde. should == solde_initial - 50
    end

    it "vide le compte lorsque le montant egale le solde" do
      @c.retirer( @c.solde )
      @c.solde. should == 0
    end

    it "signale une erreur lorsque le montant depasse le solde" do
      solde_initial = @c.solde
      lambda{ @c.retirer( 2050 ) }. should raise_error
    end
  end

end
```

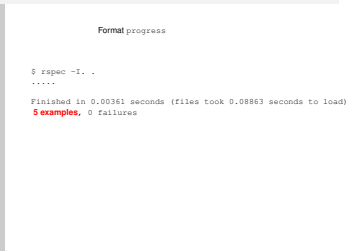



- Cet exemple pour `retirer` illustre qu'il peut y avoir , et c'est généralement le cas, plusieurs cas de tests (plusieurs `it`) pour une méthode à tester.
- Dans les versions plus récentes de RSpec, on n'utilise même plus des `should` dans les assertions, on utilise plutôt des `expect` — parce que les `should` créaient parfois des problèmes dans certains programmes — mise en oeuvre avec *«Monkey patching»*.
- Les avis sont assez partagés quant à cette nouvelle forme. Personnellement, je préfère nettement l'ancienne forme, c'est cela qui m'a attiré vers RSpec. L'autre forme ressemble plus aux anciennes assertions, même si c'est quand même différent.

Exemple RSpec

Résultats d'exécution : Format `progress` (défaut)

```
$ rspec -I. .
.....

Finished in 0.00361 seconds (files took 0.08863 seconds to load)
5 examples, 0 failures
```

- Format `progress` = format par défaut = sortie style JUnit standard
- On remarque qu'on ne parle plus de tests mais d'exemples.

Exemple RSpec

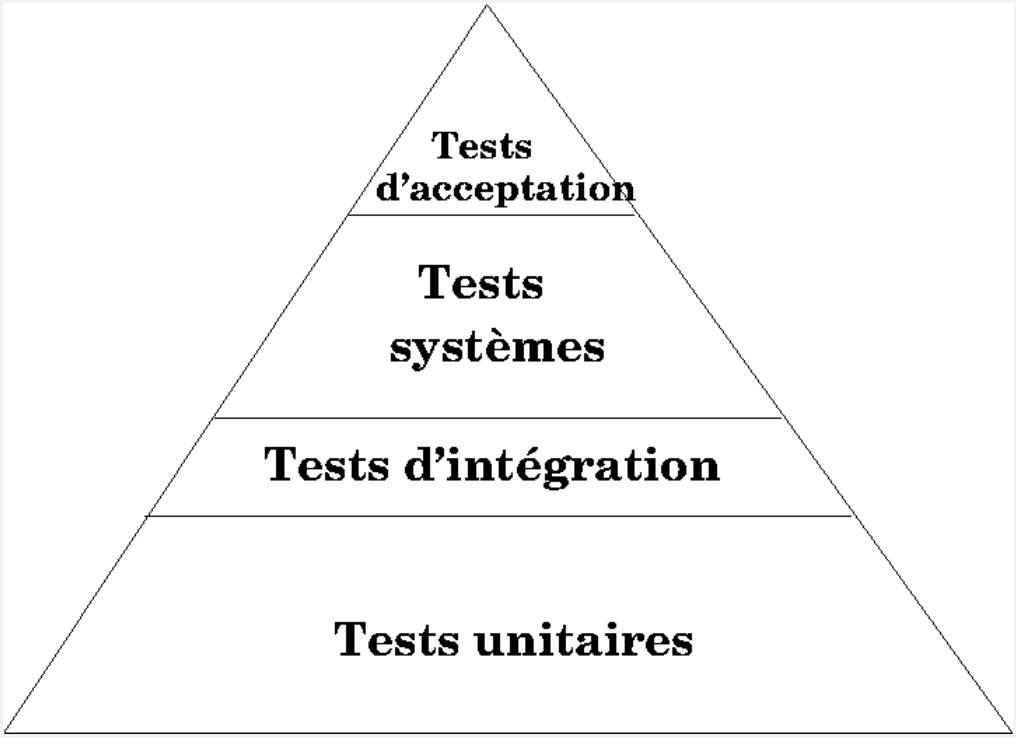
Résultats d'exécution : Format `documentation`

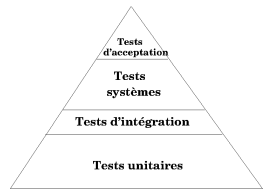
```
$  rspec -I. . --format documentation
Compte
  .new
    cree un compte avec le solde initial indique
  #deposer
    ajoute le montant indique au solde du compte
  #retirer
    deduit le montant lorsque ne depasse pas le solde
    vide le compte lorsque le montant egale le solde
    signale une erreur lorsque le montant depasse le solde

Finished in 0.00371 seconds (files took 0.08794 seconds to load)
5 examples, 0 failures
```


- Format `documentation` = donne les différents tests exécutés avec les `describe`s et les `it`s !
- Si les phrases de `describe` et des `it` sont bien formulées, cela peut parfois/souvent ressembler à une spécification informelle !
- Donc, les résulats attendus de l'exécution des tests deviennent une spécification du comportement de l'entité.

2. Que sont les tests d’acceptation ?

Différents niveaux de tests





-

Différents niveaux de tests

Tests unitaires

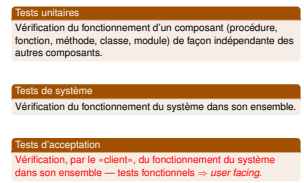
Vérification du fonctionnement d'un composant (procédure, fonction, méthode, classe, module) de façon indépendante des autres composants.

Tests de système

Vérification du fonctionnement du système dans son ensemble.

Tests d'acceptation

Vérification, par le «client», du fonctionnement du système dans son ensemble — tests fonctionnels ⇒ *user facing*.



- Tests de type alpha : Tests d'acceptation faits dans un environnement de développement
- Tests de type béta : Tests d'acceptation faits dans un environnement de production, donc une fois le système déployé... pour un nombre limité d'usagers.

Les tests d'acceptation selon RUP

Acceptance Test

The complete application (or system) is tested by end users (or representatives) for the purpose of determining readiness for deployment.

Source: «The Rational Unified Process—An Introducion (Second Edition)», *Kruchten*

- *Readiness for deployment*

⇒ Préalable pour l'intégration continue
(qu'on verra dans quelques semaines)

Acceptance Test
The complete application (or system) is tested by end users (or representatives) for the purpose of determining readiness for deployment.
Source: «The Rational Unified Process—An Introducion (Second Edition)», Kruchten
Readiness for deployment
⇒ Préalable pour l'intégration continue
(qu'on verra dans quelques semaines)

- On verra, bientôt, le lien avec l'assemblage et le déploiement de logiciels ⇒ Intégration continue !

Les tests d'acceptation selon XP

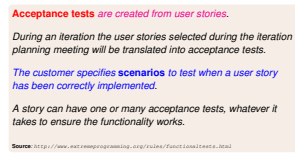
Acceptance tests *are created from user stories.*

During an iteration the user stories selected during the iteration planning meeting will be translated into acceptance tests.

The customer specifies **scenarios** *to test when a user story has been correctly implemented.*

A story can have one or many acceptance tests, whatever it takes to ensure the functionality works.

Source: `http://www.extremeprogramming.org/rules/functionaltests.html`


Acceptance tests are created from user stories.
During an iteration the user stories selected during the iteration planning meeting will be translated into acceptance tests.
The customer specifies scenarios to test when a user story has been correctly implemented.
A story can have one or many acceptance tests, whatever it takes to ensure the functionality works.
Source: http://www.extremeprogramming.org/rules/functionaltests.html

Les tests d'acceptation selon XP (suite)

Acceptance tests *are* **black box system tests**. *Each acceptance test represents some expected result from the system.*

The name **acceptance tests** *was changed from* **functional tests**. *This better reflects the intent, which is to guarantee that a customer's requirements have been met and the system is acceptable.*

Source: `http://www.extremeprogramming.org/rules/functionaltests.html`

Acceptance tests *are* **black box system tests**. *Each acceptance test represents some expected result from the system.*

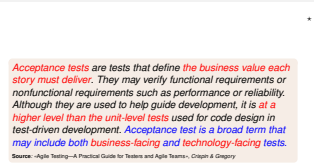
The name **acceptance tests** *was changed from* **functional tests**. *This better reflects the intent, which is to guarantee that a customer's requirements have been met and the system is acceptable.*

Source: `http://www.extremeprogramming.org/rules/functionaltests.html`

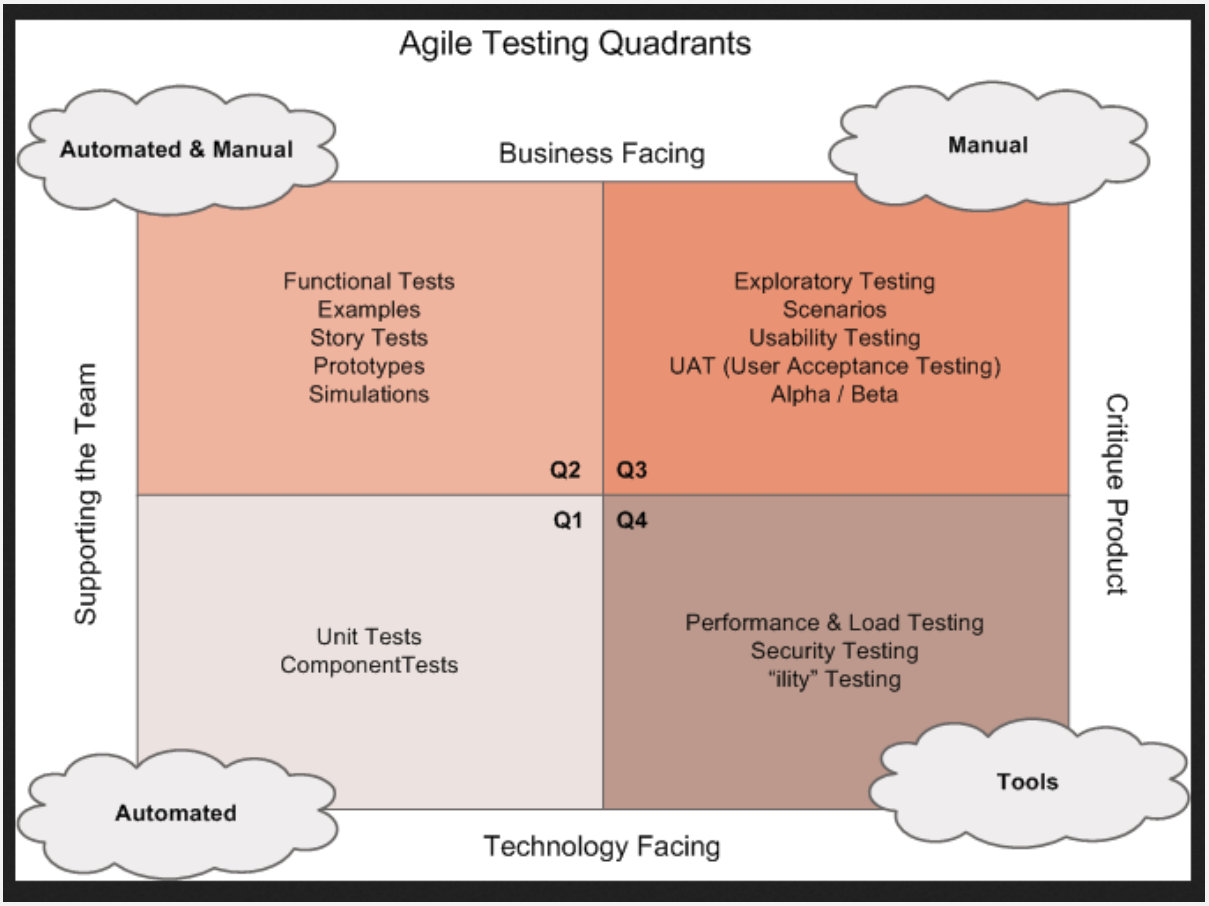
Une autre définition des tests d'acceptation ⋆

Acceptance tests are tests that define the business value each story must deliver. They may verify functional requirements or nonfunctional requirements such as performance or reliability. Although they are used to help guide development, it is at a higher level than the unit-level tests used for code design in test-driven development. Acceptance test is a broad term that may include both business-facing and technology-facing tests.

Source: «Agile Testing—A Practical Guide for Testers and Agile Teams», *Crispin & Gregory*



-

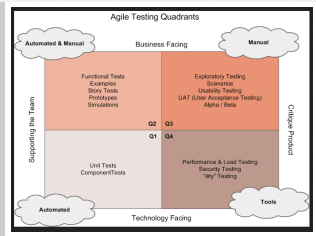


Source: `http://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/`

2016-10-14

Tests d'acceptation et BDD(*Behavior Driven Development*)

└Que sont les tests d'acceptation ?



Source: http://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/

-

3. Les tests d’acceptation et l’approche BDD

Constatation de North et Matts (2004) : L'approche BDD peut aussi s'appliquer. . . aux exigences

Toward the end of 2004, while I was describing my new found, behaviour-based vocabulary to Matts, he said, "But that's just like analysis." There was a long pause while we processed this, and then we decided to apply all of this behaviour-driven thinking to defining requirements.

Source: *Dan North,*
`http://dannorth.net/introducing-bdd/`

Toward the end of 2004, while I was describing my new found, behaviour-based vocabulary to Matts, he said, "But that's just like analysis." There was a long pause while we processed this, and then we decided to apply all of this behaviour-driven thinking to defining requirements.

Source: *Dan North,*
`http://dannorth.net/introducing-bdd/`

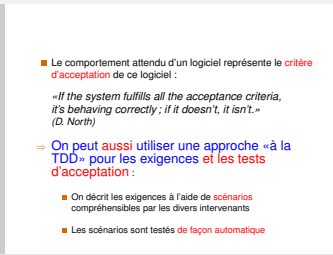
Constatation de North et Matts (2004) : L'approche BDD peut aussi s'appliquer. . . aux exigences

- Le comportement attendu d'un logiciel représente le critère d'acceptation de ce logiciel :

  «If the system fulfills all the acceptance criteria, it's behaving correctly ; if it doesn't, it isn't.»
  (D. North)

⇒ On peut aussi utiliser une approche «à la TDD» pour les exigences et les tests d'acceptation :

  - On décrit les exigences à l'aide de scénarios compréhensibles par les divers intervenants
  - Les scénarios sont testés de façon automatique



- Donc, dans un premier temps, BDD est une façon de spécifier des tests unitaires en utilisant un style différent de spécification — spécification par des exemples.
- Donc on devrait pouvoir faire du TDD aussi à l'étape d'analyse
- Avec des scénarios qui soient exécutables, comme des tests
- Qu'on pourra tester lors des tests systèmes et d'acceptation

Qu’est-ce que le BDD ?

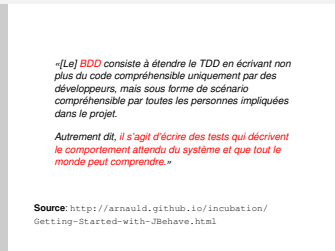
Twitter feed de D. North, oct. 2016 (`https://twitter.com/tastapod`)

Qu’est-ce que le BDD ?

«[Le] BDD consiste à étendre le TDD en écrivant non plus du code compréhensible uniquement par des développeurs, mais sous forme de scénario compréhensible par toutes les personnes impliquées dans le projet.

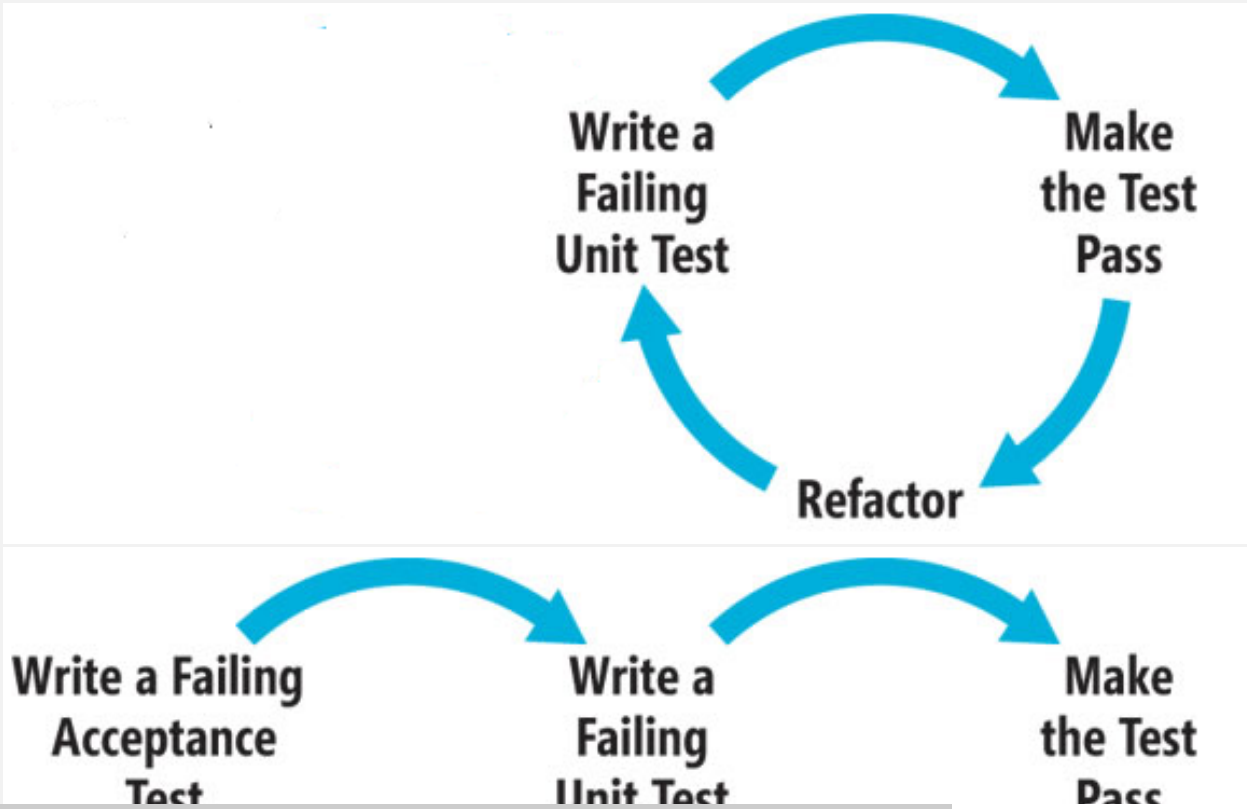
Autrement dit, il s’agit d’écrire des tests qui décrivent le comportement attendu du système et que tout le monde peut comprendre.»

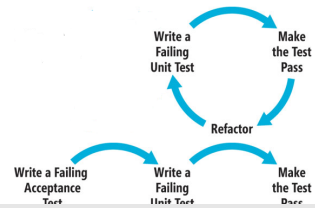
Source: `http://arnauld.github.io/incubation/Getting-Started-with-JBehave.html`



- «[G]énéralement, ces scénarios sont écrits et définis avant que l’implémentation ne commence. Ils servent à la fois à définir le besoin mais vont guider le développement en le focalisant sur la fonctionnalité décrite. Dans l’absolu, on continue à faire du TDD mais on ajoute en plus l’expression du besoin en langage naturel. Alors que le TDD garantit d’une certaine façon la qualité technique d’une implémentation, il ne garantit pas la qualité fonctionnelle. Plusieurs éléments peuvent ainsi être techniquement valides mais une fois mis ensemble ne répondent pas du tout au besoin réellement exprimé par le client. De manière un peu caricatural, le BDD va guider le développement d’une fonctionalité, tandis que le TDD guidera son implementation.

TDD vs. BDD : TDD et BDD





-

North et Matts ont dû développer un langage pour la spécification des exigences, les critères d'acceptation

If we could develop a consistent vocabulary for analysts, testers, developers, and the business, then we would be well on the way to eliminating some of the ambiguity and miscommunication that occur when technical people talk to business people.

Source*: Dan North,* `http://dannorth.net/introducing-bdd/`

If we could develop a consistent vocabulary for analysts, testers, developers, and the business, then we would be well on the way to eliminating some of the ambiguity and miscommunication that occur when technical people talk to business people.

Source: Dan North, `http://dannorth.net/introducing-bdd/`

Donc, North et Matts ont développé un *ubiquitous language* pour l'analyse

Ubiquitous Language

Ubiquitous Language = A design approach described in Eric Evans' "Domain Driven Design" (2003), which consists notably of striving to use the vocabulary of a given business domain, not only in discussions about the requirements for a software product, but in discussions of design as well and all the way into "the product's source code itself".

Source: `http://guide.agilealliance.org/guide/ubiquitous.html`

ubiquitous language

Ubiquitous Language

Ubiquitous Language = A design approach described in Eric Evans' "Domain Driven Design" (2003), which consists notably of striving to use the vocabulary of a given business domain, not only in discussions about the requirements for a software product, but in discussions of design as well and all the way into "the product's source code itself".

Source: `http://guide.agilealliance.org/guide/ubiquitous.html`

Récit utilisateur vs. Scénario utilisateur

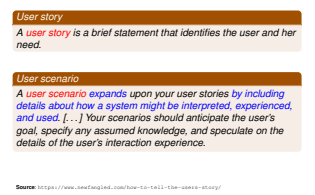
User story

A user story is a brief statement that identifies the user and her need.

User scenario

A user scenario expands upon your user stories by including details about how a system might be interpreted, experienced, and used. [. . .] Your scenarios should anticipate the user's goal, specify any assumed knowledge, and speculate on the details of the user's interaction experience.

Source: `https://www.newfangled.com/how-to-tell-the-users-story/`



-

Caractéristiques des récits utilisateurs

The general guidelines for the user stories themselves is that they must be testable, be small enough to implement in one iteration, and have business value.

Source*:* «Engineering Software as a Service»*, Fox & Patterson*

The general guidelines for the user stories themselves is that they must be testable, be small enough to implement in one iteration, and have business value.

Source*:* «Engineering Software as a Service»*, Fox & Patterson*

-

Les scénarios utilisateur décrivent les conditions d'acceptation pour un récit utilisateur

This, then, is the role of a Story. It has to be a description of a requirement and its business benefit, and a set of criteria by which we all agree that it is "done".

[The acceptance criteria] are presented as Scenarios.

Source: `http://dannorth.net/whats-in-a-story/`

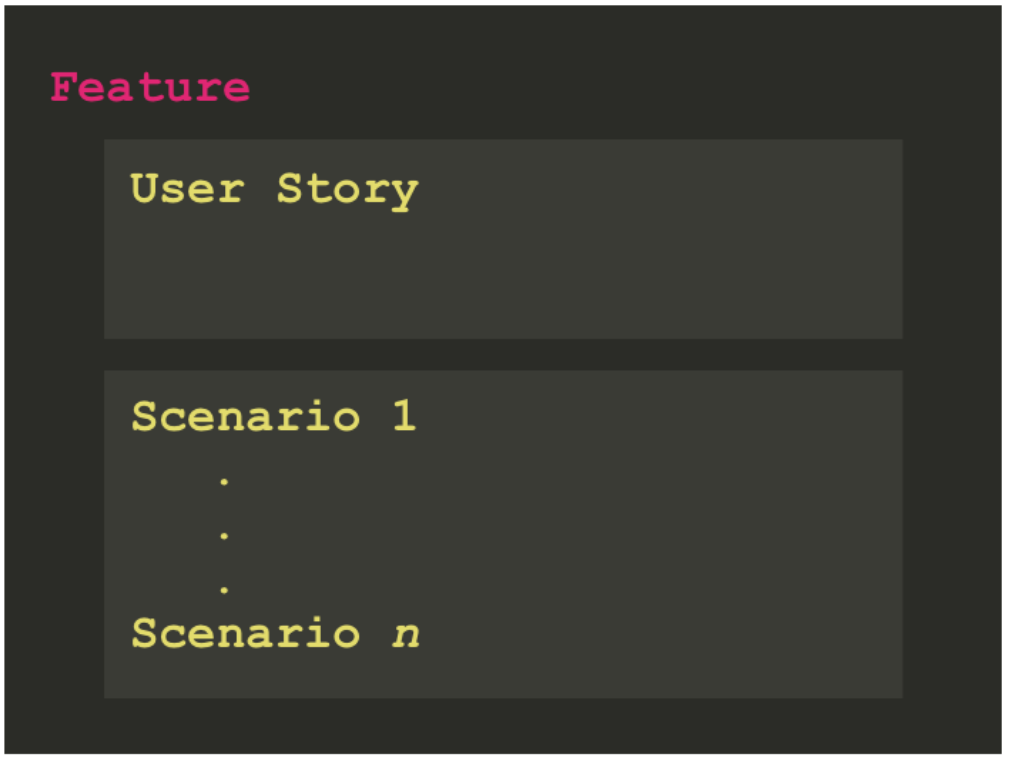
This, then, is the role of a Story. It has to be a description of a requirement and its business benefit, and a set of criteria by which we all agree that it is "done".

[The acceptance criteria] are presented as Scenarios.

Source: http://dannorth.net/whats-in-a-story/

-

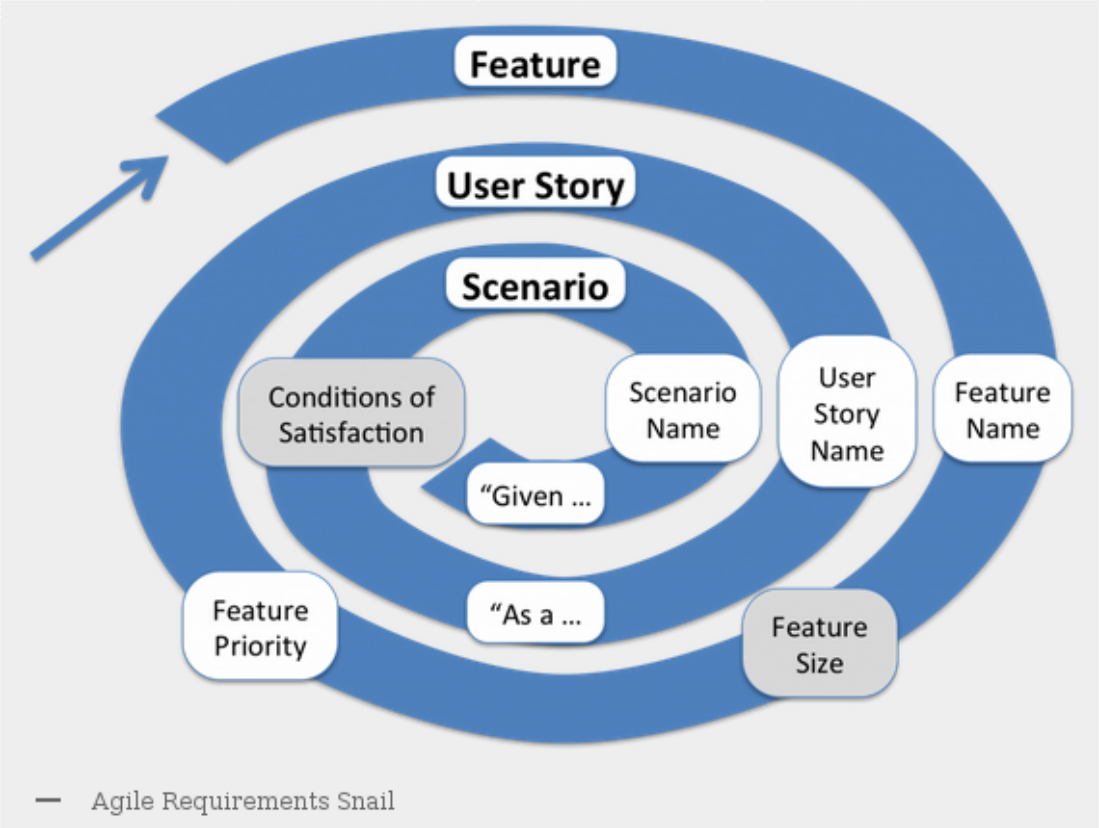
En BDD, les récits utilisateur et les scénarios utilisateur sont organisés par *feature*

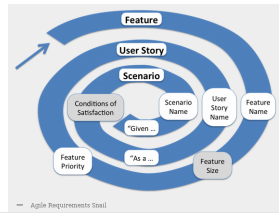


-

En BDD, les récits utilisateur et les scénarios utilisateur sont organisés par *feature*

http://itsadeliverything.com/





-

Récits utilisateurs : Un gabarit «standard»

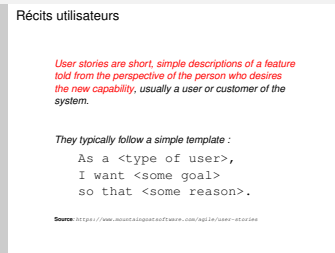
(style connextra)

User stories are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system.

They typically follow a simple template :

```
As a <type of user>,
I want <some goal>
so that <some reason>.
```

Source: *https://www.mountaingoatsoftware.com/agile/user-stories*



-

Scénarios utilisateurs : Le gabarit de North & Matts

```
Scenario: name of the scenario
 Given some initial context
 When an event occurs
 Then ensure some outcomes
```

North et Matts, avec `JBehave`, ont conçu une façon d'exécuter des scénarios exprimés dans ce langage

Scénarios utilisateurs

Scenario: name of the scenario
Given some initial context
When an event occurs
Then ensure some outcomes

North et Matts, avec JBehave, ont conçu une façon d'exécuter des scénarios exprimés dans ce langage

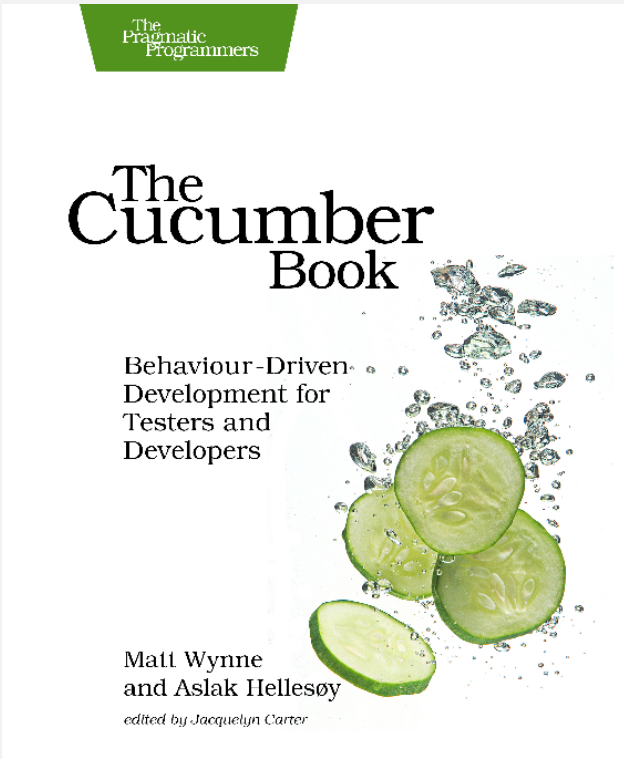
-

4. Des outils pour le BDD

4.1 cucumber

-

Cucumber (Hellesoy, 2008)

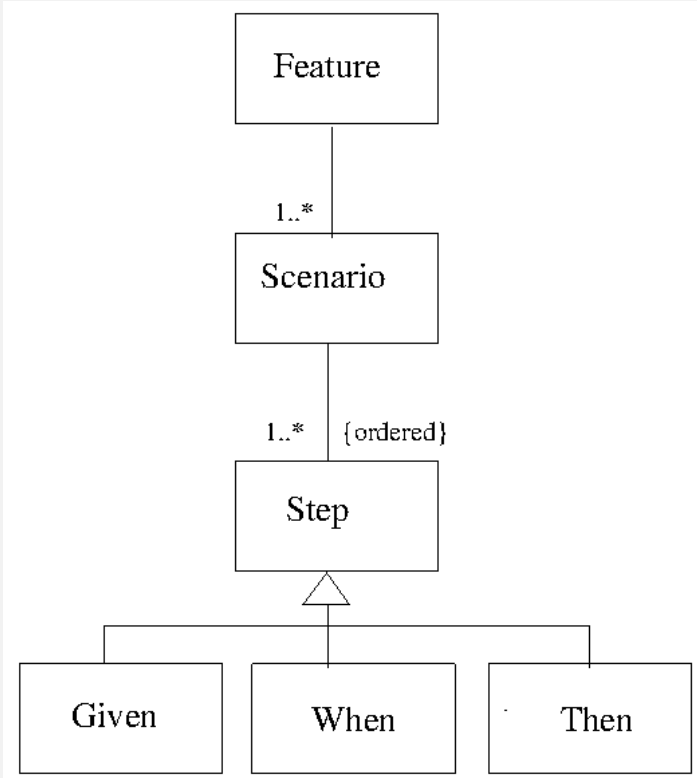


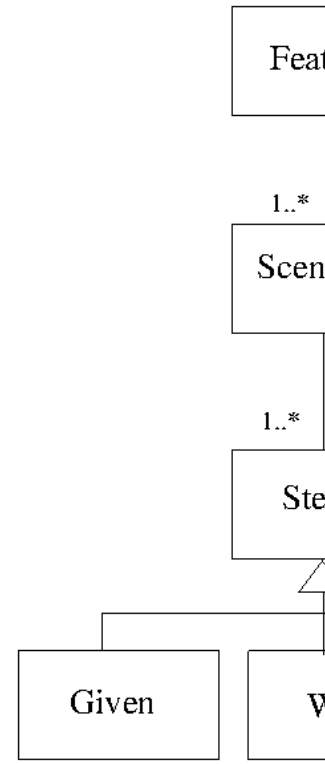
Pour la spécification et l'exécution des tests d'acceptation !

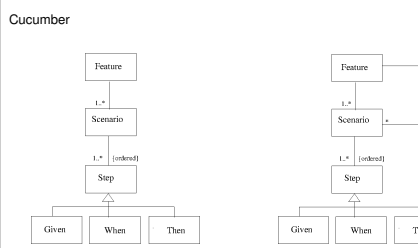
Cucumber
Pour la spécification et l'exécution des tests d'acceptation !

- Cette idée de scénario exécutable a alors été reprise, en Ruby, par Hellesoy, dans un outil appelé `cucumber`, que je vais vous présenter.
- C'est cet outil que je présente en premier, car c'est celui avec lequel je suis plus familier.

Cucumber : Principe général de fonctionnement





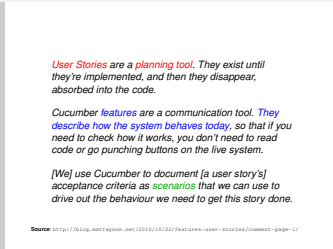


-

Les *features* de `cucumber` vs. les *user stories*

> *User Stories are a planning tool. They exist until they're implemented, and then they disappear, absorbed into the code.*
>
> *Cucumber features are a communication tool. They describe how the system behaves today, so that if you need to check how it works, you don't need to read code or go punching buttons on the live system.*
>
> *[We] use Cucumber to document [a user story's] acceptance criteria as scenarios that we can use to drive out the behaviour we need to get this story done.*

Source: `http://blog.mattwynne.net/2010/10/22/features-user-stories/comment-page-1/`


User Stories are a planning tool. They exist until they're implemented, and then they disappear, absorbed into the code.

Cucumber features are a communication tool. They describe how the system behaves today, so that if you need to check how it works, you don't need to read code or go punching buttons on the live system.

[We] use Cucumber to document [a user story's] acceptance criteria as scenarios that we can use to drive out the behaviour we need to get this story done.

Source: http://blog.mattwynne.net/2010/10/22/features-user-stories/comment-page-1/

- Aussi : «*User Stories are a great way to plan your work. You can take a big hairy requirement and break it down into chunks that are small enough to work on without anyone freaking out. When you've crumbled up your big hairy requirement into little user story chunks, you can pick and choose which chunk to build first, and even drop some chunks altogether when you realise they're not that important. Great stuff.*»

Les trois sortes de `step`s qui composent un `scenario` Cucumber

Given

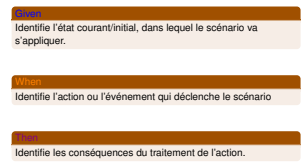
Identifie l'état courant/initial, dans lequel le scénario va s'appliquer.

When

Identifie l'action ou l'événement qui déclenche le scénario

Then

Identifie les conséquences du traitement de l'action.



-

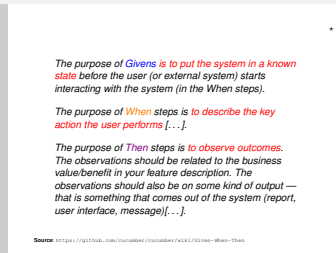
Les trois sortes de `steps` qui composent un `scenario` Cucumber

The purpose of Givens is to put the system in a known state before the user (or external system) starts interacting with the system (in the When steps).

The purpose of When steps is to describe the key action the user performs [. . .].

The purpose of Then steps is to observe outcomes. The observations should be related to the business value/benefit in your feature description. The observations should also be on some kind of output — that is something that comes out of the system (report, user interface, message)[. . .].

Source: `https://github.com/cucumber/cucumber/wiki/Given-When-Then`



-

Les trois sortes de `steps` qui composent un `scenario` Cucumber

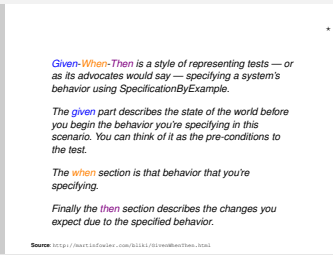
Given-When-Then is a style of representing tests — or as its advocates would say — specifying a system's behavior using SpecificationByExample.

The given part describes the state of the world before you begin the behavior you're specifying in this scenario. You can think of it as the pre-conditions to the test.

The when section is that behavior that you're specifying.

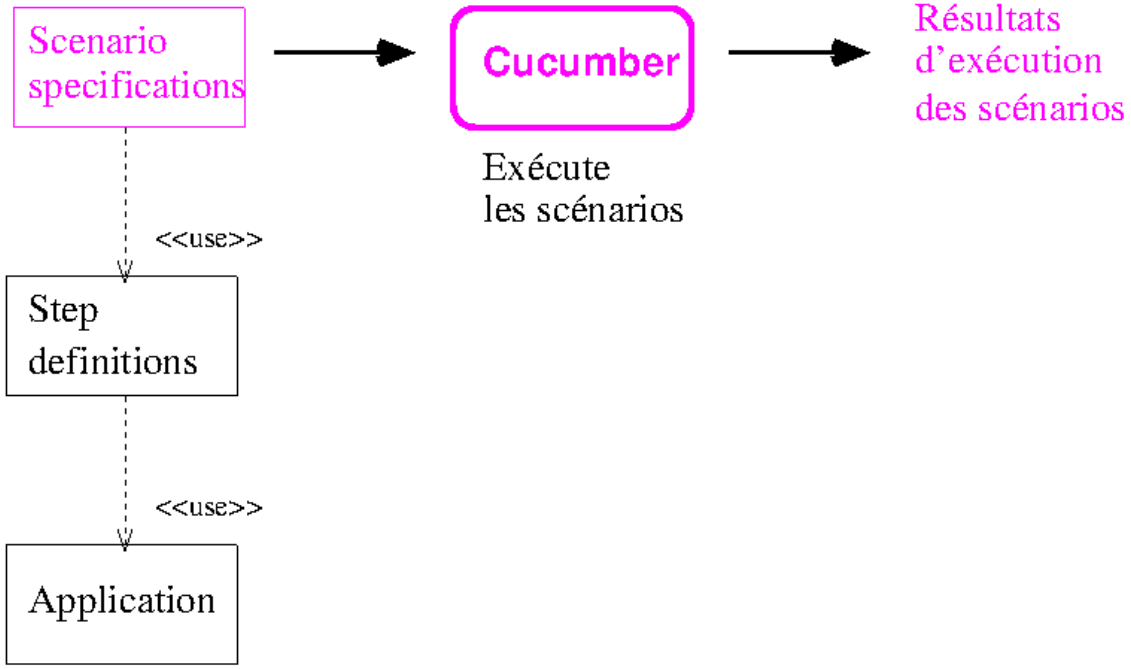
Finally the then section describes the changes you expect due to the specified behavior.

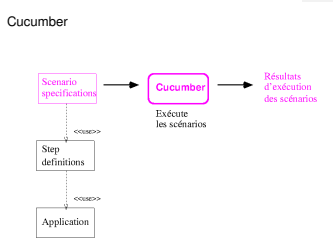
Source: `http://martinfowler.com/bliki/GivenWhenThen.html`


Given-When-Then is a style of representing tests — or as its advocates would say — specifying a system's behavior using SpecificationByExample.
The given part describes the state of the world before you begin the behavior you're specifying in this scenario. You can think of it as the pre-conditions to the test.
The when section is that behavior that you're specifying.
Finally the then section describes the changes you expect due to the specified behavior.
Source: http://martinfowler.com/bliki/GivenWhenThen.html

-

Cucumber : Principe général de fonctionnement

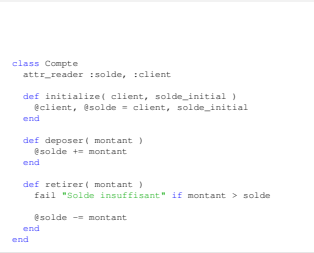




-

Exemple Ruby avec `cucumber` et `gherkin` : Classe à tester

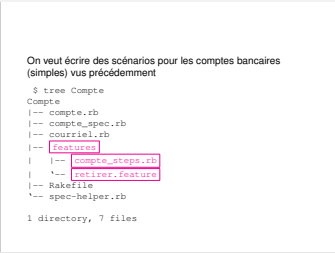
```ruby
class Compte
  attr_reader :solde, :client

  def initialize( client, solde_initial )
    @client, @solde = client, solde_initial
  end

  def deposer( montant )
    @solde += montant
  end

  def retirer( montant )
    fail "Solde insuffisant" if montant > solde

    @solde -= montant
  end
end
```

Exemple Ruby avec `cucumber` et `gherkin` : Scénarios d'utilisation d'un compte bancaire

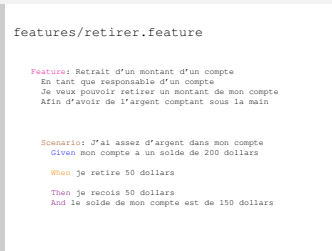
On veut écrire des scénarios pour les comptes bancaires (simples) vus précédemment

```
 $ tree Compte
Compte
|-- compte.rb
|-- compte_spec.rb
|-- courriel.rb
|-- features
|   |-- compte_steps.rb
|   `-- retirer.feature
|-- Rakefile
`-- spec-helper.rb

1 directory, 7 files
```



On veut écrire des scénarios pour les comptes bancaires (simples) vus précédemment
$ tree Compte
Compte
|-- compte.rb
|-- compte_spec.rb
|-- courriel.rb
|-- features
| |-- compte_steps.rb
| `-- retirer.feature
|-- Rakefile
`-- spec-helper.rb
1 directory, 7 files

Exemple Ruby avec `cucumber` et `gherkin` : `features/retirer.feature` (1)

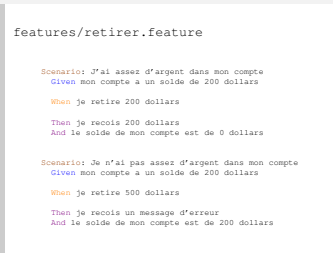
```
Feature: Retrait d'un montant d'un compte
  En tant que responsable d'un compte
  Je veux pouvoir retirer un montant de mon compte
  Afin d'avoir de l'argent comptant sous la main


  Scenario: J'ai assez d'argent dans mon compte
    Given mon compte a un solde de 200 dollars

    When je retire 50 dollars

    Then je recois 50 dollars
    And le solde de mon compte est de 150 dollars
```

- Voici un exemple pour la version simplifiée de la banque.
- `cucumber` = l'outil d'exécution des scénarios
- `gherkin` = le langage de description/spécification des scénarios
- La description de la «*feature*», au tout début, avant les scénarios, est arbitraire (texte non analysé par l'outil). Toutefois, la suggestion/convention est d'utiliser gabarit dit *Connextra format* pour la description des récits utilisateurs (*user stories*).

Exemple Ruby avec `cucumber` et `gherkin` : `features/retirer.feature` (2)

```
Scenario: J'ai assez d'argent dans mon compte
  Given mon compte a un solde de 200 dollars

  When je retire 200 dollars

  Then je recois 200 dollars
  And le solde de mon compte est de 0 dollars


Scenario: Je n'ai pas assez d'argent dans mon compte
  Given mon compte a un solde de 200 dollars

  When je retire 500 dollars

  Then je recois un message d'erreur
  And le solde de mon compte est de 200 dollars
```

-

Exécution initiale… avec des étapes *pending* (suite)

```
<<~/SeminaireTBDDD/Compte@MacBook>> $ cucumber
Feature: Retrait d'un montant d'un compte
  En tant que responsable d'un compte
  Je veux pouvoir retirer un montant de mon compte
  Afin d'avoir de l'argent comptant sous la main

  Scenario: J'ai assez d'argent dans mon compte   # features/retirer.feature:6
    Given mon compte a un solde de 200 dollars    # features/retirer.feature:7
    When je retire 50 dollars                     # features/retirer.feature:9
    Then je recois 50 dollars                     # features/retirer.feature:11
    And le solde de mon compte est de 150 dollars # features/retirer.feature:12

  Scenario: J'ai assez d'argent dans mon compte # features/retirer.feature:15
    Given mon compte a un solde de 200 dollars  # features/retirer.feature:16
    When je retire 200 dollars                  # features/retirer.feature:18
    Then je recois 200 dollars                  # features/retirer.feature:20
    And le solde de mon compte est de 0 dollars # features/retirer.feature:21

  Scenario: Je n'ai pas assez d'argent dans mon compte # features/retirer.feature:24
    Given mon compte a un solde de 200 dollars         # features/retirer.feature:25
    When je retire 500 dollars                         # features/retirer.feature:27
    Then je recois un message d'erreur                 # features/retirer.feature:29
    And le solde de mon compte est de 200 dollars      # features/retirer.feature:30

3 scenarios (3 undefined)
12 steps (12 undefined)
0m0.006s

You can implement step definitions for undefined steps with these snippets:
```

```
You can im

Given(/^mo
  pending
end

When(/^je
  pending
end

Then(/^je
  pending
end

Then(/^le
  pending
end

Then(/^je
  pending
end
```

2016-10-14

Tests d'acceptation et BDD(*Behavior Driven Development*)
└Des outils pour le BDD
└`cucumber`
└Exécution initiale… avec des étapes

- L'exécution initiale de ces scénarios, sans mises en oeuvre des diverses étapes, donnerait alors un résultat comme celui-ci.
- On remarque que cela dit «*3 scenarios*» et «*12 steps*». Chaque étape correspond à un *Given*, *When* ou *Then*.
- Dans l'état initial, ces étapes ne sont pas encore mises en oeuvre, et `cucumber` nous donne des suggestions quant à ce qu'il faut faire pour démarrer leur mise en oeuvre.

Mise en oeuvrè avec RSpec (très simplifiée) : `features/compte_steps.rb`

```ruby
NB = Transform /^\d+$/ do |nb| nb.to_i end

Given(/^mon compte a un solde de (#{NB}) dollars$/) do |montant|
  @c = Compte.new( "MOI", montant )
end

When(/^je retire (#{NB}) dollars$/) do |montant|
  @montant_recu = nil
  begin
    @c.retirer montant
    @montant_recu = montant
  rescue Exception => e
    @erreur = e
  end
end
```


features/compte_steps.rb

NB = Transform /^\d+$/ do |nb| nb.to_i end

Given(/^mon compte a un solde de (#{NB}) dollars$/) do |montant|
@c = Compte.new("MOI", montant)
end

When(/^je retire (#{NB}) dollars$/) do |montant|
@montant_recu = nil
begin
@c.retirer montant
@montant_recu = montant
rescue Exception => e
@erreur = e
end
end

- Voici maintenant une mise en oeuvre très simplifiée des diverses étapes, qui permet d'exécuter les divers scénarios de tests avec succès.
- Dans un premier temps, voici les Given et les When, donc les pré-conditions.
- Qu'est-ce que je vais faire pour satisfaire cet antécédent. . . je vais créer un compte avec un certain solde — donc c'est comme le *setup* d'un test unitaire.

Mise en oeuvrè avec RSpec (très simplifiée) : `features/compte_steps.rb` (suite)

```
Then(/^le solde de mon compte est de (#{NB}) dollars$/)
      do |montant|
  expect( @c.solde ).to eq montant
end

Then(/^je recois (#{NB}) dollars$/) do |montant|
  expect( @montant_recu ).to eq montant
end

Then(/^je recois un message d'erreur$/) do
  expect( @erreur ).not_to be_nil
end
```


features/compte_steps.rb

Then(/^le solde de mon compte est de (#{NB}) dollars$/)
do |montant|
expect(@c.solde).to eq montant
end

Then(/^je recois (#{NB}) dollars$/) do |montant|
expect(@montant_recu).to eq montant
end

Then(/^je recois un message d'erreur$/) do
expect(@erreur).not_to be_nil
end

- Voici ensuite les Then, donc les post-conditions.

Mise en oeuvrè avec RSpec (très simplifiée) : `features/compte_steps.rb` (suite)

```
Then(/^le solde de mon compte est de (#{NB}) dollars$/)
      do |montant|
  expect( @c.solde ).to eq montant
end

Then(/^je recois (#{NB}) dollars$/) do |montant|
  expect( @montant_recu ).to eq montant
end

Then(/^je recois un message d'erreur$/) do
  expect( @erreur ).not_to be_nil
end
```


features/compte_steps.rb

Then(/^le solde de mon compte est de (#{NB}) dollars$/)
do |montant|
expect(@c.solde).to eq montant
end

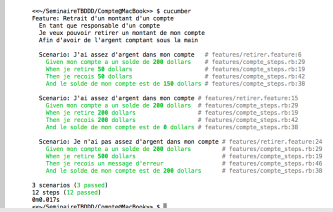
Then(/^je recois (#{NB}) dollars$/) do |montant|
expect(@montant_recu).to eq montant
end

Then(/^je recois un message d'erreur$/) do
expect(@erreur).not_to be_nil
end

- Je vous signale que les attentes sont exprimées dans le nouveau style aussi suggéré par RSpec, donc avec des `expect` plutôt qu'avec des `should`.

Exécution. . . après avoir finalisé les «étapes»

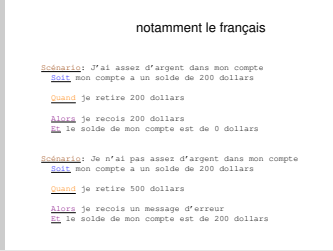
```
<<~/SeminaireTBDDD/Compte@MacBook>> $ cucumber
Feature: Retrait d'un montant d'un compte
  En tant que responsable d'un compte
  Je veux pouvoir retirer un montant de mon compte
  Afin d'avoir de l'argent comptant sous la main

  Scenario: J'ai assez d'argent dans mon compte   # features/retirer.feature:6
    Given mon compte a un solde de 200 dollars    # features/compte_steps.rb:29
    When je retire 50 dollars                     # features/compte_steps.rb:19
    Then je recois 50 dollars                     # features/compte_steps.rb:42
    And le solde de mon compte est de 150 dollars # features/compte_steps.rb:38

  Scenario: J'ai assez d'argent dans mon compte # features/retirer.feature:15
    Given mon compte a un solde de 200 dollars  # features/compte_steps.rb:29
    When je retire 200 dollars                  # features/compte_steps.rb:19
    Then je recois 200 dollars                  # features/compte_steps.rb:42
    And le solde de mon compte est de 0 dollars # features/compte_steps.rb:38

  Scenario: Je n'ai pas assez d'argent dans mon compte # features/retirer.feature:24
    Given mon compte a un solde de 200 dollars         # features/compte_steps.rb:29
    When je retire 500 dollars                         # features/compte_steps.rb:19
    Then je recois un message d'erreur                 # features/compte_steps.rb:46
    And le solde de mon compte est de 200 dollars      # features/compte_steps.rb:38

3 scenarios (3 passed)
12 steps (12 passed)
0m0.017s
<<~/SeminaireTBDDD/Compte@MacBook>> $
```

- Voici donc les résultats d'exécution une fois qu'on a défini ces étapes.

Le DSL de `cucumber`, `gherkin`, supporte de nombreux langages, notamment le français

```
Scénario: J'ai assez d'argent dans mon compte
  Soit mon compte a un solde de 200 dollars

  Quand je retire 200 dollars

  Alors je recois 200 dollars
  Et le solde de mon compte est de 0 dollars


Scénario: Je n'ai pas assez d'argent dans mon compte
  Soit mon compte a un solde de 200 dollars

  Quand je retire 500 dollars

  Alors je recois un message d'erreur
  Et le solde de mon compte est de 200 dollars
```

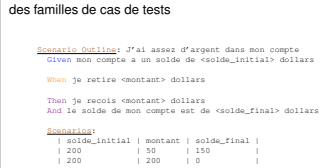



- Le langage `gherkin` supporte une très grande quantité de langues : 37 en date de Nov. 2013 !

Le DSL de `cucumber`, `gherkin`, permet de définir des familles de cas de tests

```
Scenario Outline: J'ai assez d'argent dans mon compte
  Given mon compte a un solde de <solde_initial> dollars

  When je retire <montant> dollars

  Then je recois <montant> dollars
  And le solde de mon compte est de <solde_final> dollars

  Scenarios:
    | solde_initial | montant | solde_final |
    | 200           | 50      | 150         |
    | 200           | 200     | 0           |
```


des familles de cas de tests



-

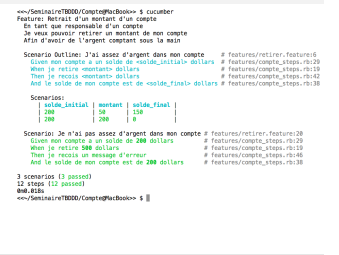
```
<<~/SeminaireTBDDD/Compte@MacBook>> $ cucumber
Feature: Retrait d'un montant d'un compte
  En tant que responsable d'un compte
  Je veux pouvoir retirer un montant de mon compte
  Afin d'avoir de l'argent comptant sous la main

  Scenario Outline: J'ai assez d'argent dans mon compte     # features/retirer.feature:6
    Given mon compte a un solde de <solde_initial> dollars  # features/compte_steps.rb:29
    When je retire <montant> dollars                        # features/compte_steps.rb:19
    Then je recois <montant> dollars                        # features/compte_steps.rb:42
    And le solde de mon compte est de <solde_final> dollars # features/compte_steps.rb:38

    Scenarios:
      | solde_initial | montant | solde_final |
      | 200           | 50      | 150         |
      | 200           | 200     | 0           |

  Scenario: Je n'ai pas assez d'argent dans mon compte # features/retirer.feature:20
    Given mon compte a un solde de 200 dollars         # features/compte_steps.rb:29
    When je retire 500 dollars                         # features/compte_steps.rb:19
    Then je recois un message d'erreur                 # features/compte_steps.rb:46
    And le solde de mon compte est de 200 dollars      # features/compte_steps.rb:38

3 scenarios (3 passed)
12 steps (12 passed)
0m0.018s
<<~/SeminaireTBDDD/Compte@MacBook>> $
```

2016-10-14

Tests d'acceptation et BDD(*Behavior Driven Development*)

└Des outils pour le BDD

└cucumber



-

4.2 JBehave

-

Quelques caractéristiques de `JBehave`

- *Open source*
- Mise en oeuvre entièrement Java
- Utilise des annotations pour associer une méthode Java à une étape

- *Open source*
- Mise en oeuvre entièrement Java
- Utilise des annotations pour associer une méthode Java à une étape

-

Quelques caractéristiques de JBehave

JBehave peut être exécuté de différentes façons

Source: http://jbehave.org/

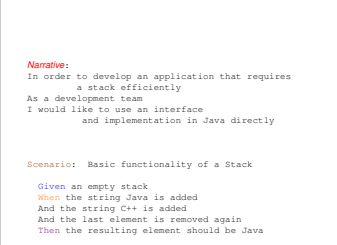
-

La description des scénarios ≈ comme cucumber

Tiré de https://blog.codecentric.de/en/2011/03/automated-acceptance-testing-using-jbehave/

Narrative:
```
In order to develop an application that requires
         a stack efficiently
As a development team
I would like to use an interface
          and implementation in Java directly
```

```
Scenario:  Basic functionality of a Stack

  Given an empty stack
  When the string Java is added
  And the string C++ is added
  And the last element is removed again
  Then the resulting element should be Java
```

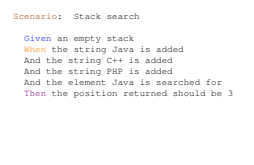



-

La description des scénarios ≈ comme cucumber

Tiré de https://blog.codecentric.de/en/2011/03/automated-acceptance-testing-using-jbehave/

```
Scenario:  Stack search

  Given an empty stack
  When the string Java is added
  And the string C++ is added
  And the string PHP is added
  And the element Java is searched for
  Then the position returned should be 3
```

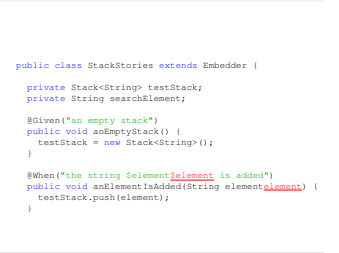



-

La mise en oeuvre des étapes = Java avec annotations

Tiré de https://blog.codecentric.de/en/2011/03/automated-acceptance-testing-using-jbehave/

```
public class StackStories extends Embedder {

  private Stack<String> testStack;
  private String searchElement;

  @Given("an empty stack")
  public void anEmptyStack() {
    testStack = new Stack<String>();
  }

  @When("the string $element$element is added")
  public void anElementIsAdded(String elementelement) {
    testStack.push(element);
  }
```

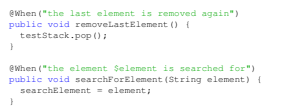



-

La mise en oeuvre des étapes = Java avec annotations

Tiré de https://blog.codecentric.de/en/2011/03/automated-acceptance-testing-using-jbehave/

```
@When("the last element is removed again")
public void removeLastElement() {
  testStack.pop();
}

@When("the element $element is searched for")
public void searchForElement(String element) {
  searchElement = element;
}
```

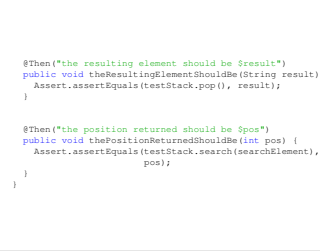



-

La mise en oeuvre des étapes = Java avec annotations

Tiré de https://blog.codecentric.de/en/2011/03/automated-acceptance-testing-using-jbehave/

```
  @Then("the resulting element should be $result")
  public void theResultingElementShouldBe(String result) {
    Assert.assertEquals(testStack.pop(), result);
  }


  @Then("the position returned should be $pos")
  public void thePositionReturnedShouldBe(int pos) {
    Assert.assertEquals(testStack.search(searchElement),
                        pos);
  }
}
```

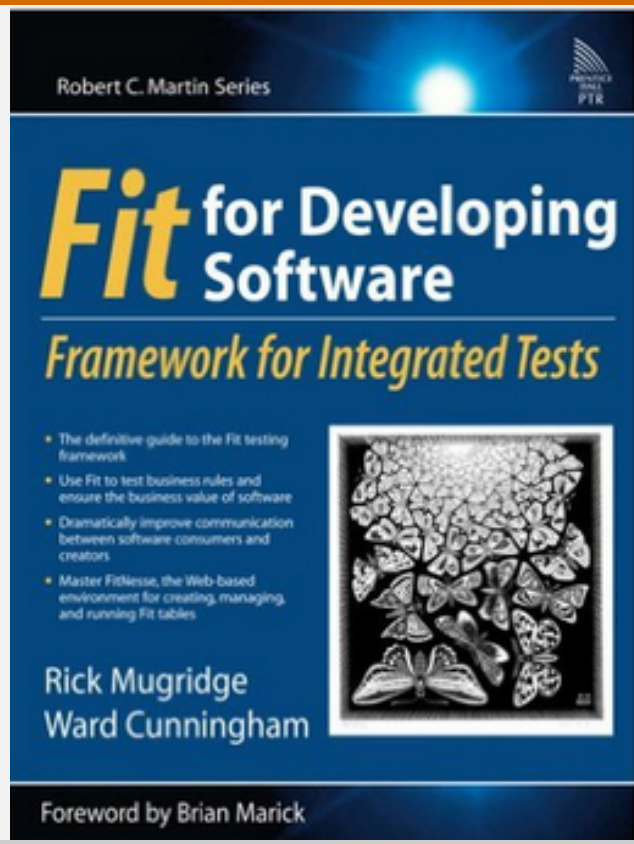
-

4.3 Fit

-

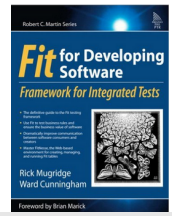
Fit : *Framework for Integrated Tests*

W. Cunningham = Inventeur/concepteur du premier wiki



2016-10-14

Tests d'acceptation et BDD(*Behavior Driven Development*)
└Des outils pour le BDD
└Fit
└Fit : *Framework for Integrated Tests*



-

Fit : *Framework for Integrated Tests*

Framework for Integrated Test, or “Fit”, is an open-source tool for automated customer tests. It integrates the work of customers, analysts, testers, and developers.

Source: `https://en.wikipedia.org/wiki/Framework_for_integrated_test`

Framework for Integrated Test, or “Fit”, is an open-source tool for automated customer tests. It integrates the work of customers, analysts, testers, and developers.

Source: `https://en.wikipedia.org/wiki/Framework_for_integrated_test`

-

Fit : *Framework for Integrated Tests*

Customers provide examples of how their software should work. Those examples are then connected to the software with programmer-written test fixtures and automatically checked for correctness.

The customers' examples are formatted in tables and saved as HTML using ordinary business tools such as Microsoft Excel. When Fit checks the document, it creates a copy and colors the tables **green**, **red**, *and* **yellow** *according to whether the software behaved as expected.*

Source: `https://en.wikipedia.org/wiki/Framework_for_integrated_test`

Customers provide examples of how their software should work. Those examples are then connected to the software with programmer-written test fixtures and automatically checked for correctness.

The customers' examples are formatted in tables and saved as HTML using ordinary business tools such as Microsoft Excel. When Fit checks the document, it creates a copy and colors the tables **green**, **red**, *and* **yellow** *according to whether the software behaved as expected.*

Source: `https://en.wikipedia.org/wiki/Framework_for_integrated_test`

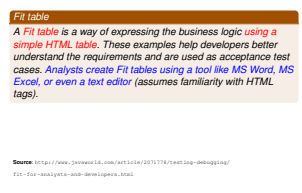
-

Les concepts de base de Fit

Fit table

A Fit table is a way of expressing the business logic using a simple HTML table. These examples help developers better understand the requirements and are used as acceptance test cases. Analysts create Fit tables using a tool like MS Word, MS Excel, or even a text editor (assumes familiarity with HTML tags).

Source: `http://www.javaworld.com/article/2071778/testing-debugging/fit-for-analysts-and-developers.html`



-

Les concepts de base de Fit

Fixture

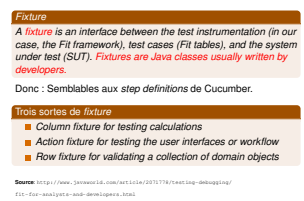
A fixture is an interface between the test instrumentation (in our case, the Fit framework), test cases (Fit tables), and the system under test (SUT). Fixtures are Java classes usually written by developers.

Donc : Semblables aux *step definitions* de Cucumber.

Trois sortes de *fixture*

- *Column fixture for testing calculations*
- *Action fixture for testing the user interfaces or workflow*
- *Row fixture for validating a collection of domain objects*

Source: `http://www.javaworld.com/article/2071778/testing-debugging/fit-for-analysts-and-developers.html`



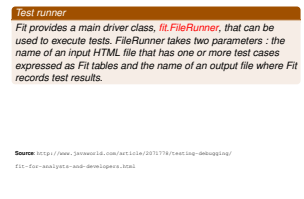
•

Les concepts de base de Fit

Test runner

Fit provides a main driver class, fit.FileRunner, that can be used to execute tests. FileRunner takes two parameters : the name of an input HTML file that has one or more test cases expressed as Fit tables and the name of an output file where Fit records test results.

Source: `http://www.javaworld.com/article/2071778/testing-debugging/fit-for-analysts-and-developers.html`



-

Les concepts de base de Fit

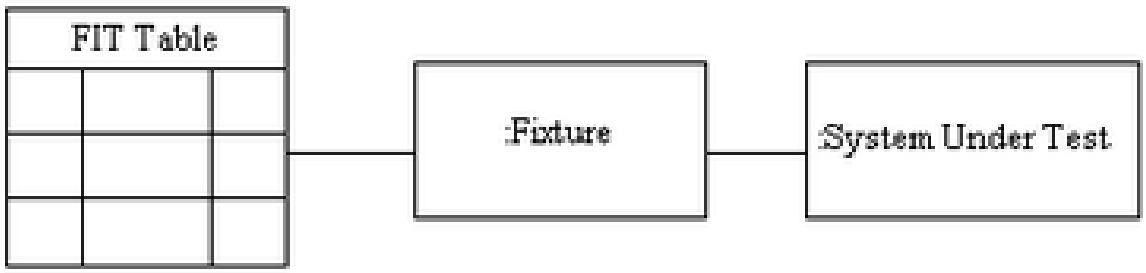


Figure 1. Relationship between Fit table, fixture, and SUT

Source: `http://www.javaworld.com/article/2071778/testing-debugging/fit-for-analysts-and-developers.html`

FIT Table
:Fixture
System Under Test
Figure 1. Relationship between Fit table, fixture, and SUT
Source: http://www.javaworld.com/article/2071778/testing-debugging/fit-for-analysts-and-developers.html

-

Processus d'utilisation de Fit

On identifie les récits utilisateurs

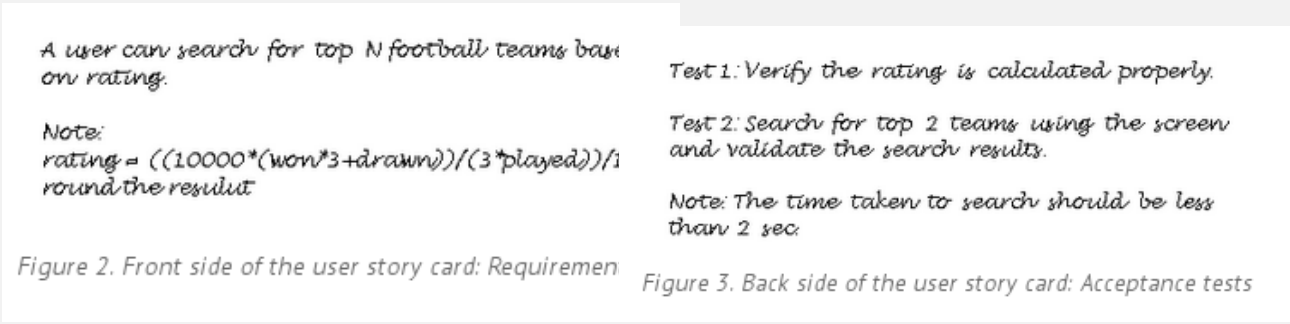


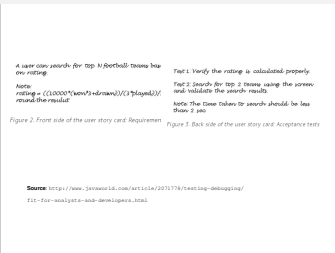
Figure 2. Front side of the user story card: Requiremen

Figure 3. Back side of the user story card: Acceptance tests

Source: `http://www.javaworld.com/article/2071778/testing-debugging/fit-for-analysts-and-developers.html`

2016-10-14

Tests d'acceptation et BDD(*Behavior Driven Development*)
└Des outils pour le BDD
└Fit
└Processus d'utilisation de Fit



-

Processus d'utilisation de Fit

On crée les tables d'exemples, par exemple, dans un fichier Excel

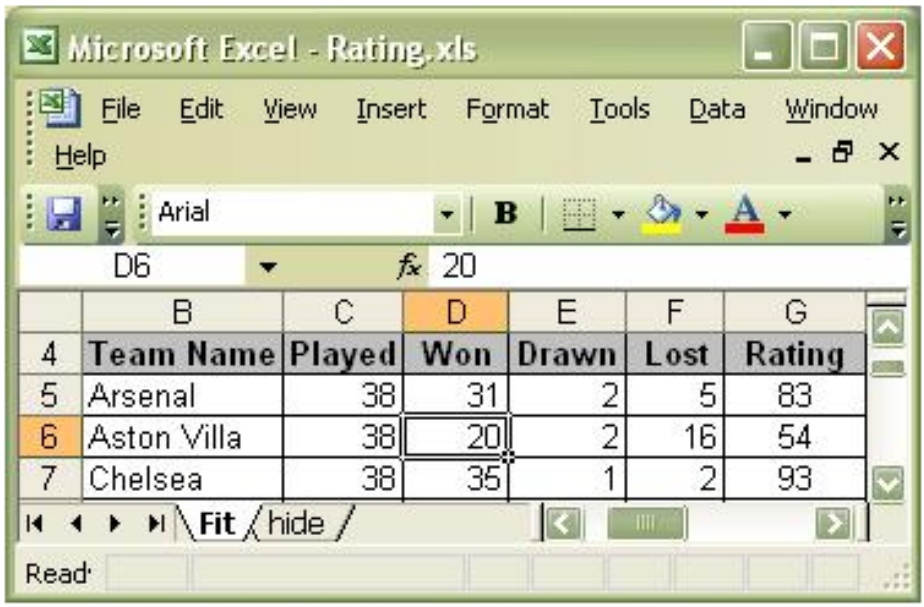
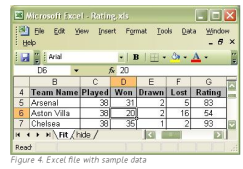


Figure 4. Excel file with sample data

Source: `http://www.javaworld.com/article/2071778/testing-debugging/fit-for-analysts-and-developers.html`

2016-10-14

Tests d'acceptation et BDD(*Behavior Driven Development*)
└Des outils pour le BDD
└Fit
└Processus d'utilisation de Fit



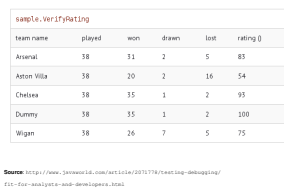
-

Processus d'utilisation de Fit

Des fichiers HTML sont générés à partir des tables des fichiers Excel

sample.VerifyRating					
team name	played	won	drawn	lost	rating ()
Arsenal	38	31	2	5	83
Aston Villa	38	20	2	16	54
Chelsea	38	35	1	2	93
Dummy	38	35	1	2	100
Wigan	38	26	7	5	75

Source: `http://www.javaworld.com/article/2071778/testing-debugging/fit-for-analysts-and-developers.html`



-

Processus d'utilisation de Fit

On écrit le code Java qui fait le lien avec les exemples

```java
package sample;
import fit.ColumnFixture;

public class VerifyRating extends ColumnFixture {
  public String teamName;
  public int played;
  public int won;
  public int drawn;
  public int lost;
  Team team = null;

  public long rating(){
    team = new Team(teamName,played,won,drawn,lost);
    return team.rating;
  }
}
```

Source: http://www.javaworld.com/article/2071778/testing-debugging/

package sample;
import fit.ColumnFixture;

public class VerifyRating extends ColumnFixture {
public String teamName;
public int played;
public int won;
public int drawn;
public int lost;
Team team = null;

public long rating(){
team = new Team(teamName,played,won,drawn,lost);
return team.rating;
}
}

Source: http://www.javaworld.com/article/2071778/testing-debugging/

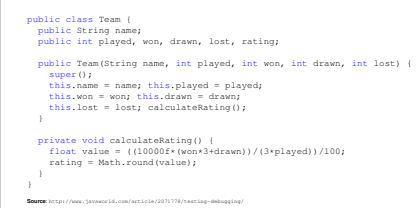
-

Processus d'utilisation de Fit

On écrit le code Java qui met en oeuvre le modèle d'affaire

```
public class Team {
  public String name;
  public int played, won, drawn, lost, rating;

  public Team(String name, int played, int won, int drawn, i
    super();
    this.name = name; this.played = played;
    this.won = won; this.drawn = drawn;
    this.lost = lost; calculateRating();
  }

  private void calculateRating() {
    float value = ((10000f*(won*3+drawn))/(3*played))/100;
    rating = Math.round(value);
  }
}
```

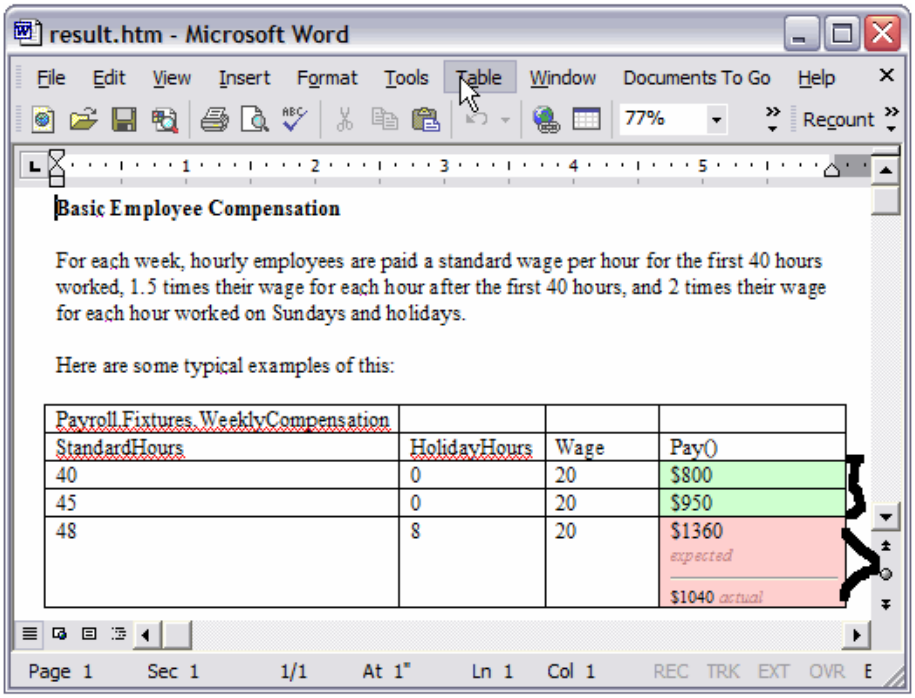
Source: http://www.javaworld.com/article/2071778/testing-debugging/



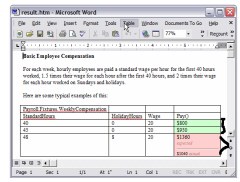
-

Processus d'utilisation de Fit

On lance le *test runner* pour vérifier que tous les exemples sont satisfaits



Source: `http://fit.c2.com/wiki.cgi?IntroductionToFit`



- Pas le même exemple que les précédents. Illustre seulement les cellules vertes et rouges.

4.4 D’autres outils pour le BDD

-

De nombreux autres outils sont disponibles pour l'approche BDD

Cucumber-JVM

Cucumber-JVM is a Cucumber implementation for the most popular JVM languages.

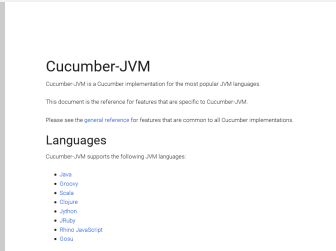
This document is the reference for features that are specific to Cucumber-JVM.

Please see the general reference for features that are common to all Cucumber implementations.

Languages

Cucumber-JVM supports the following JVM languages:

- Java
- Groovy
- Scala
- Clojure
- Jython
- JRuby
- Rhino JavaScript
- Gosu



-

De nombreux autres outils sont disponibles pour l'approche BDD

specflow
Cucumber for .NET

-

De nombreux autres outils sont disponibles pour l'approche BDD

behat
A php framework for autotesting your business expectations.

-

De nombreux autres outils sont disponibles pour l'approche BDD

-

5. D’autres outils pour les tests d’acceptation

Fait : L'interface personne–machine de nombreux systèmes repose sur l'utilisation de pages Web et de fureteurs

Comment peut-on interagir, dans un programme de tests, avec des sites Web ?

Comment peut-on interagir, dans un programme de tests, avec des sites Web ?

A-t-on besoin d'un fureteur (*browser*) pour interagir avec des sites Web ?

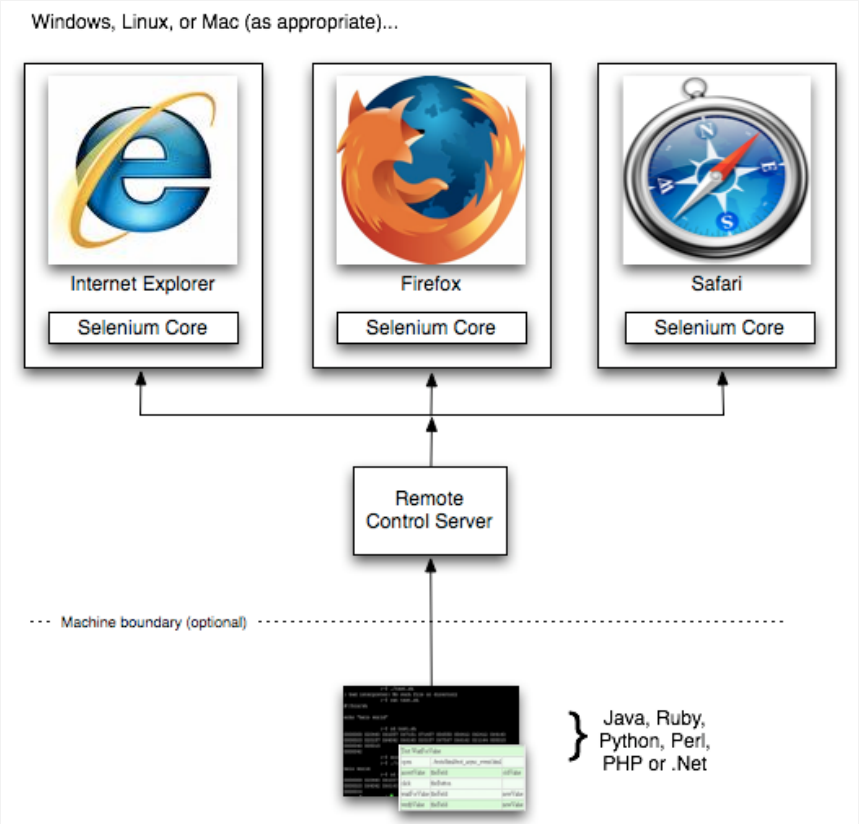
A-t-on besoin d'un fureteur (*browser*) pour interagir avec des sites Web ?

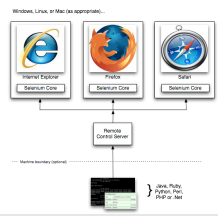
5.1 Contrôleurs de fureteurs vs. fureteurs «sans tête»

-

Certains outils permettent de contrôler l'exécution d'un «vrai» fureteur = *Browser drivers*

`http://www.seleniumhq.org/selenium-rc.png`





-

Exemple : Tests unitaires de l'application Web d'Oto

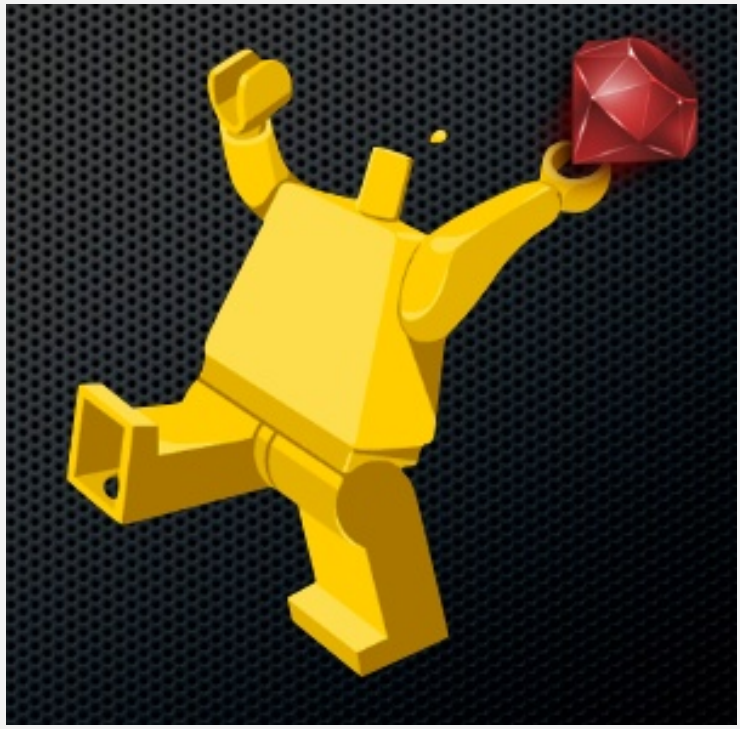
Vidéo illustrant une *partie* de l'exécution des tests unitaires de l'application Web d'Oto avec Firefox :
`http://www.labunix.uqam.ca/~tremblay/MGL7460/Materiel/video-tests-oto.mov`

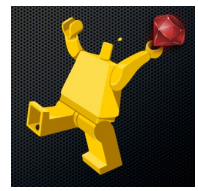
Vidéo illustrant une *partie* de l'exécution des tests unitaires de l'application Web d'Oto avec Firefox :
`http://www.labunix.uqam.ca/~tremblay/MGL7460/Materiel/video-tests-oto.mov`

-

D’autres outils mettent en oeuvre des fureteurs. . . sans GUI ! = *Headless browser*

http://techyworks.blogspot.ca/2014/08/headless-browser-testing-using-selenium.html





-

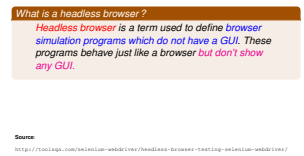
Headless browser

What is a headless browser ?

Headless browser is a term used to define *browser simulation programs which do not have a GUI*. These programs behave just like a browser *but don't show any GUI.*

Source:

`http://toolsqa.com/selenium-webdriver/headless-browser-testing-selenium-webdriver/`



-

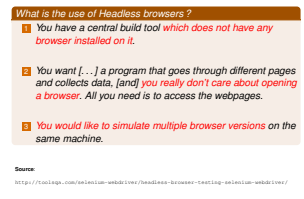
Headless browser

What is the use of Headless browsers ?

1. *You have a central build tool which does not have any browser installed on it.*
2. *You want [. . .] a program that goes through different pages and collects data, [and] you really don't care about opening a browser. All you need is to access the webpages.*
3. *You would like to simulate multiple browser versions on the same machine.*

Source:

`http://toolsqa.com/selenium-webdriver/headless-browser-testing-selenium-webdriver/`



-

5.2 Langages de description d'interactions avec des sites Web

-

Question : Comment peut-on interagir, dans un programme de tests, avec des sites Web ?

Réponse = En utilisant un langage qui permet de décrire les interactions avec les sites Web

Question : Comment peut-on interagir, dans un programme de tests, avec des sites Web ?

Réponse = En utilisant un langage qui permet de décrire les interactions avec les sites Web

-

Capybara

-

`Capybara` est un outil Ruby pour interagir avec des sites Web, avec ou sans fureteur

Capybara

Capybara is an integration testing tool for rack based web applications. It simulates how a user would interact with a website.

-

`Capybara` est un outil Ruby pour interagir avec des sites Web, avec ou sans fureteur *

Capybara

Capybara is a tool that Ruby on Rails developers mostly use for testing their web applications. This tool, however, can be also used to automate boring/repeating/long running tasks on the web or scraping information from web sites that were not kind enough to provide API.

2016-10-14

Tests d'acceptation et BDD(*Behavior Driven Development*)
└D'autres outils pour les tests d'acceptation
└Langages de description d'interactions avec des sites Web

-

`Capybara` définit un DSL pour décrire des interactions avec des sites Web

Navigation

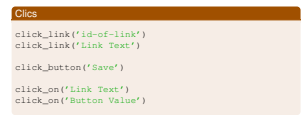
```
visit('/projects')
visit('http://oto.uqam.ca')
```


-

`Capybara` définit un DSL pour décrire des interactions avec des sites Web

Clics

```
click_link('id-of-link')
click_link('Link Text')

click_button('Save')

click_on('Link Text')
click_on('Button Value')
```

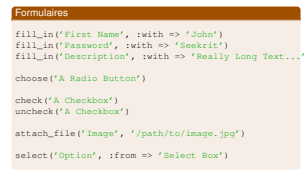



-

`Capybara` définit un DSL pour décrire des interactions avec des sites Web

Formulaires

```
fill_in('First Name', :with => 'John')
fill_in('Password', :with => 'Seekrit')
fill_in('Description', :with => 'Really Long Text...')

choose('A Radio Button')

check('A Checkbox')
uncheck('A Checkbox')

attach_file('Image', '/path/to/image.jpg')

select('Option', :from => 'Select Box')
```

-

`Capybara` définit un DSL pour décrire des interactions avec des sites Web

Requêtes

```
page.has_selector?('table tr')
page.has_selector?(:xpath, '//table/tr')

page.has_xpath?('//table/tr')
page.has_css?('table tr.foo')

page.has_content?('foo')
```


-

`Capybara` définit un DSL pour décrire des interactions avec des sites Web

Recherches

```
find_field('First Name').value

find_link('Hello', :visible => :all).visible?

find_button('Send').click

find(:xpath, '//table/tr').click
find('#overlay').find('h1').click
```


-

Divers *drivers* peuvent être utilisés, avec un «vrai» fureteur ou avec un fureteur «sans tête»

Driver Rack

- Driver par défaut de Capybara
- Interagit avec un serveur sans GUI (*Headless browser*)
- Ne peut pas exécuter de JavaScript

Driver Selenium

- Peut contrôler un vrai fureteur : Voir la vidéo plus loin
- Ne peut pas exécuter de JavaScript

Example

Les tests unitaires de l'application Web pour l'outil de correction Oto sont décrits avec le DSL de `Capybara`, et exécutés avec le *driver* Selenium pour `Firefox`.

-

Autre exemple d'utilisation de Capybara : Une application pour gérer des prêts de livres

Voir plus loin !

-

Selenium WebDriver

-

Des DSL semblables sont disponibles pour d'autres outils et d'autres langages

Exemple : `WebDriver` de Selenium pour Java

WebDriver

```
WebDriver driver
  = new FirefoxDriver();

driver.get( "http://localhost.8080/#/welcome");

driver.findElement(By.name("email"))
      .sendKeys("jane.smith@acme.com");
driver.findElement(By.name("password"))
      .sendKeys("s3cr3t");
driver.findElement(By.id("signin"))
      .click();

WebElement welcomeMsg
  = driver.findElement(By.id("welcome-message"));

assertThat(welcomeMsg.text()).isEqualTo("Welcome Jane");
```

Source: «*BDD in action*», Smart

-

5.3 Outils de capture et de réexécution

-

Question : Comment peut-on interagir, dans un programme de tests, avec des sites Web ?

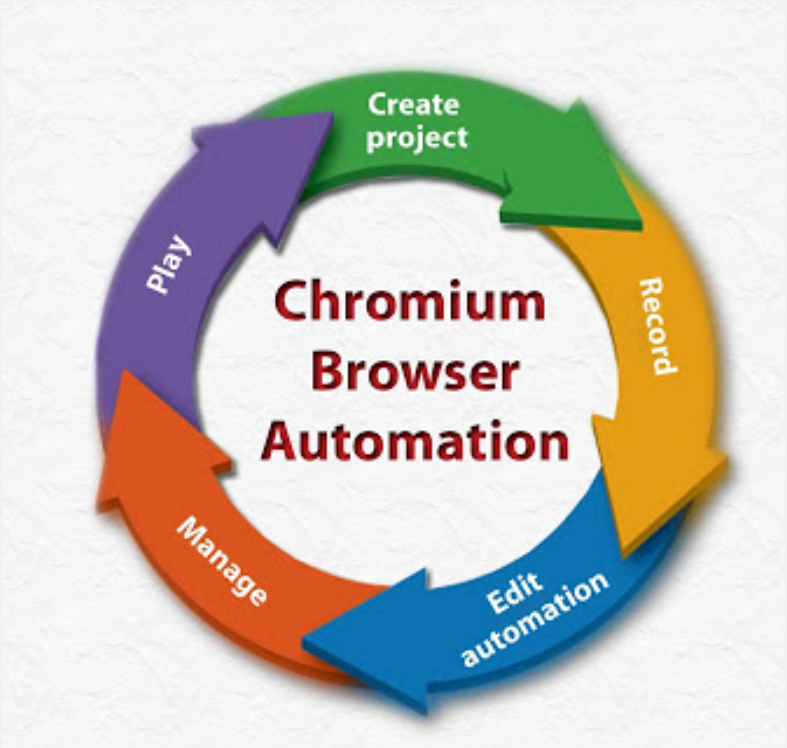
Réponse = En «enregistrant» des interactions faites de façon manuelle, puis en «rejouant» les actions enregistrées

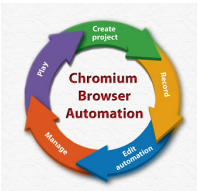
Question : Comment peut-on interagir, dans un programme de tests, avec des sites Web ?

Réponse = En «enregistrant» des interactions faites de façon manuelle, puis en «rejouant» les actions enregistrées

-

Certains outils permettent de capturer les interactions avec un fureteur, puis de les reproduire





-

Selenium

-

Which part of Selenium is appropriate for me?

Selenium WebDriver

If you want to

- create robust, browser-based regression automation suites and tests
- scale and distribute scripts across many environments

Selenium IDE

If you want to

- create quick bug reproduction scripts
- create scripts to aid in automation-aided exploratory testing

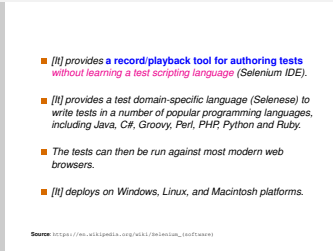
Source: `http://www.seleniumhq.org/`

-

Selenium est un outil pour interagir avec des sites Web, dans divers langages et environnements

- *[It] provides* **a record/playback tool for authoring tests** *without learning a test scripting language (Selenium IDE).*
- *[It] provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including Java, C#, Groovy, Perl, PHP, Python and Ruby.*
- *The tests can then be run against most modern web browsers.*
- *[It] deploys on Windows, Linux, and Macintosh platforms.*

Source: `https://en.wikipedia.org/wiki/Selenium_(software)`



• It is open-source software, released under the Apache 2.0 license, and can be downloaded and used without charge.

5.4 Outils en ligne de commandes

-

Question : Comment peut-on interagir, dans un programme de tests, avec des sites Web ?

Réponse = En exécutant, via un script, des commandes au niveau de la «ligne de commandes»

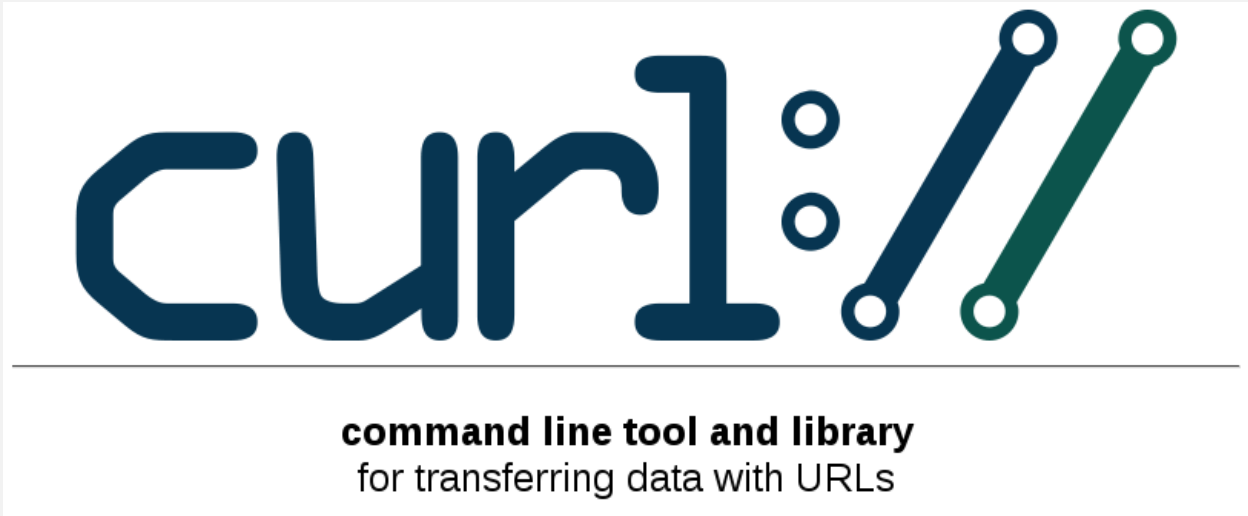
Question : Comment peut-on interagir, dans un programme de tests, avec des sites Web ?

Réponse = En exécutant, via un script, des commandes au niveau de la «ligne de commandes»

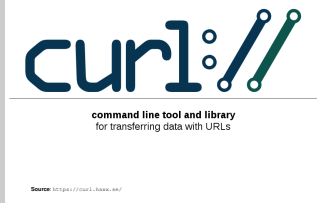
-

curl

-



Source: `https://curl.haxx.se/`



-

`curl` permet de faire des accès Web en mode «ligne de commandes»

curl is a tool to transfer data from or to a server, using one of the supported protocols (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, TELNET and TFTP).

The command is designed to work without user interaction.

Source: `man curl`

⇒ Parfait pour utilisation dans des scripts de tests

curl is a tool to transfer data from or to a server, using one of the supported protocols (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, TELNET and TFTP).

The command is designed to work without user interaction.

Source: `man curl`

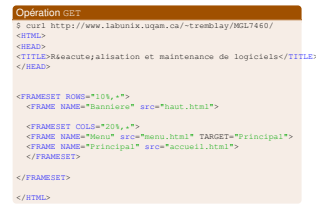
⇒ Parfait pour utilisation dans des scripts de tests

-

L'opération par défaut de `curl` est `GET`

Opération `GET`

```
$ curl http://www.labunix.uqam.ca/~tremblay/MGL7460/
<HTML>
<HEAD>
<TITLE>R&eacute;alisation et maintenance de logiciels</TITLE>
</HEAD>


<FRAMESET ROWS="10%,*">
  <FRAME NAME="Banniere" src="haut.html">

  <FRAMESET COLS="20%,*">
  <FRAME NAME="Menu" src="menu.html" TARGET="Principal">
  <FRAME NAME="Principal" src="accueil.html">
  </FRAMESET>

</FRAMESET>

</HTML>
```

-

Mais toute autre opération HTTP peut aussi être exécutée

Opération `POST` pour un formulaire avec champs à remplir

```
$ curl --cookie ....\
     -F "champ1=..."\
     -F "champ2=..."\
     -F "submit=..."\
     http://www.labunix.uqam.ca/...
```


-

Quand un tel outil peut-il être utile ?

Quand un tel outil peut-il être utile ?

Exemple : Utilisation de `curl` pour l'application Web d'Oto — tests de stress et de concurrence

Script de haut niveau

Pendant plusieurs rondes, on lance plusieurs utilisateurs (10) qui vont accéder l'application «en même temps» (concurrence) :

Extraits du script `run-tests-curl`

```
for (( i=1; i<=$NB_RONDES; i++ )); do
  for (( j=0; j<$NB_TESTEURS; j++ )); do
    verifier-puis-rendre-tp testeur$j ... &
  done
  wait
  ...
done
```

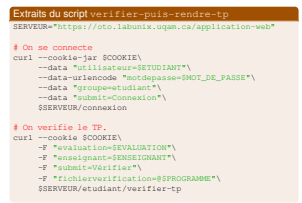

- Raisons de ces tests : L'application Web semblait bien fonctionner, sauf parfois lors de la remise des laboratoires, donc lorsqu'il y avait une période de deux heures allouées pour faire un travail et que les remises se faisaient alors «presque toutes en même temps» à la fin de la période de remise.
- Je soupçonnais évidemment une «situation de compétition» (*race condition*), mais je n'arrivais pas à voir où et, surtout, je n'arrivais pas, avec des tests manuels ou avec des tests Capybara, à reproduire le problème. Par contre, l'utilisation de ces scripts `curl` m'a permis de réussir à reproduire le bogue, ce qui m'a permis ensuite de mieux cerner où était le problème puis de le régler — ajout de 3 lignes de code, en utilisant un verrou pour protéger un accès à une ressource partagée.

Exemple : Utilisation de `curl` pour l'application Web d'Oto — tests de stress et de concurrence

Les appels `curl` dans `verifier-puis-rendre-tp`

Extraits du script `verifier-puis-rendre-tp`

```
SERVEUR="https://oto.labunix.uqam.ca/application-web"

# On se connecte
curl --cookie-jar $COOKIE\
     --data "utilisateur=$ETUDIANT"\
     --data-urlencode "motdepasse=$MOT_DE_PASSE"\
     --data "groupe=etudiant"\
     --data "submit=Connexion"\
     $SERVEUR/connexion

# On verifie le TP.
curl --cookie $COOKIE\
     -F "evaluation=$EVALUATION"\
     -F "enseignant=$ENSEIGNANT"\
     -F "submit=Vérifier"\
     -F "fichierverification=@$PROGRAMME"\
     $SERVEUR/etudiant/verifier-tp
```

-

Exemple : Utilisation de `curl` pour l'application Web d'Oto — tests de stress et de concurrence

Les appels `curl` dans `verifier-puis-rendre-tp`

Extraits du script `verifier-puis-rendre-tp` (suite)

```
# On rend le TP
curl --cookie $COOKIE\
     -F "boite=$BOITE"\
     -F "enseignant=$ENSEIGNANT"\
     -F "submit=Rendre"\
     -F "equipe=$EQUIPE"\
     -F "fichier1=@$PROGRAMME"\
     $SERVEUR/etudiant/rendre-tp

# On se deconnecte
curl --cookie $COOKIE\
     $SERVEUR/deconnexion
```


-

5.5 *How much web testing do you really need ?*

-

Extraits de «*BDD in Action*»

Web tests clearly have their uses. But you rarely need to test every aspect of a system using web tests, and doing so is generally not a good idea. In fact, in a typical BDD project, a significant proportion of automated acceptance tests will be implemented as non-web tests.

[. . .]

Many automated acceptance criteria, particularly those related to business rules or calculations, are more effectively done using the application code rather than via the user-interface, as non-web tests can test specific business rules more quickly and more precisely then an end-to-end web test.

Web tests clearly have their uses. But you rarely need to test every aspect of a system using web tests, and doing so is generally not a good idea. In fact, in a typical BDD project, a significant proportion of automated acceptance tests will be implemented as non-web tests.
[. . .]
Many automated acceptance criteria, particularly those related to business rules or calculations, are more effectively done using the application code rather than via the user-interface, as non-web tests can test specific business rules more quickly and more precisely then an end-to-end web test.

-

Extraits de «*BDD in Action*» (suite)

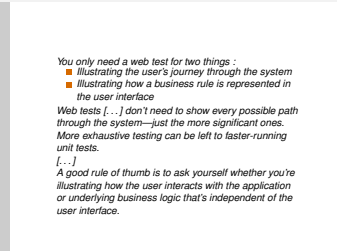
You only need a web test for two things :

- *Illustrating the user's journey through the system*
- *Illustrating how a business rule is represented in the user interface*

Web tests [. . .] don't need to show every possible path through the system—just the more significant ones. More exhaustive testing can be left to faster-running unit tests.

[. . .]

A good rule of thumb is to ask yourself whether you're illustrating how the user interacts with the application or underlying business logic that's independent of the user interface.



-

6. Un exemple plus détaillé avec `cucumber:biblio`

Le logiciel `biblio` : Première version C (≈ 1998)

`biblio`

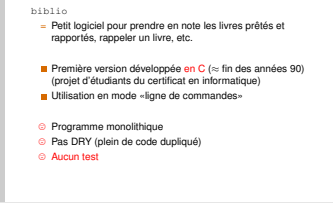
= Petit logiciel pour prendre en note les livres prêtés et rapportés, rappeler un livre, etc.

- Première version développée en C (≈ fin des années 90) (projet d'étudiants du certificat en informatique)
- Utilisation en mode «ligne de commandes»

☹ Programme monolithique

☹ Pas DRY (plein de code dupliqué)

☹ Aucun test



•

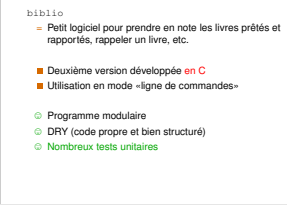
Le logiciel `biblio` : Deuxième version C (≈ 2005)

`biblio`

= Petit logiciel pour prendre en note les livres prêtés et rapportés, rappeler un livre, etc.

- Deuxième version développée en C
- Utilisation en mode «ligne de commandes»

☺ Programme modulaire
☺ DRY (code propre et bien structuré)
☺ Nombreux tests unitaires



•

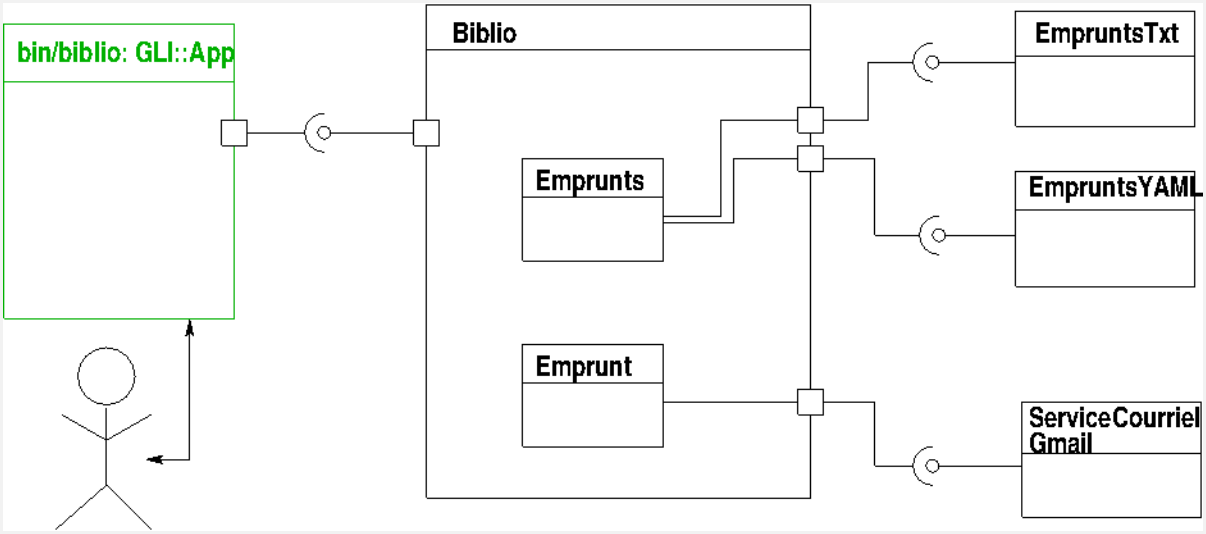
Le logiciel `biblio` : Version Ruby (2014)

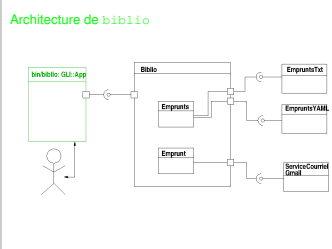
`biblio`

- Nouvelle version développée en Ruby
- Avec tests
  - Tests unitaires (`rspec`)
  - Tests d'acceptation (`cucumber`)
- Deux modes d'utilisation
  - Mode ligne de commande (`gli`)
  - Mode Web (`Ruby on Rails`)

-

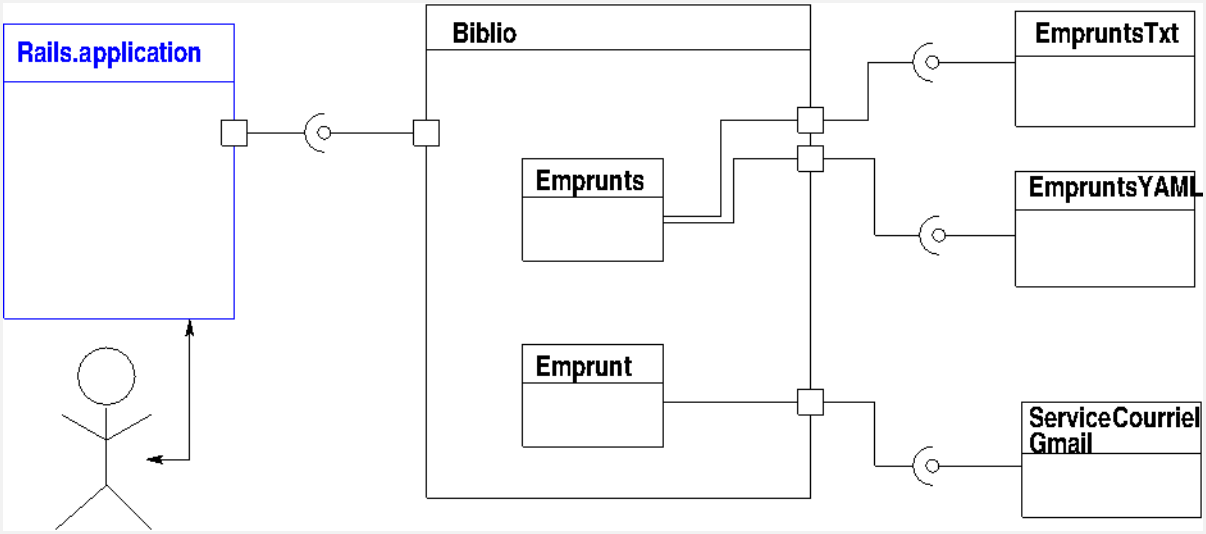
Architecture de `biblio`

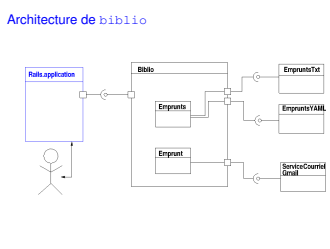




- Voici donc ce que ça donne dans le contexte de mon application `biblio` pour la gestion de prêts de livres.

Architecture de `biblio`





• Quand on veut changer d'interface personne-machine pour une application Web, il suffit essentiellement de définir un nouveau composant, qui va utiliser les autres composants déjà définis.

6.1 Spécification de `biblio`

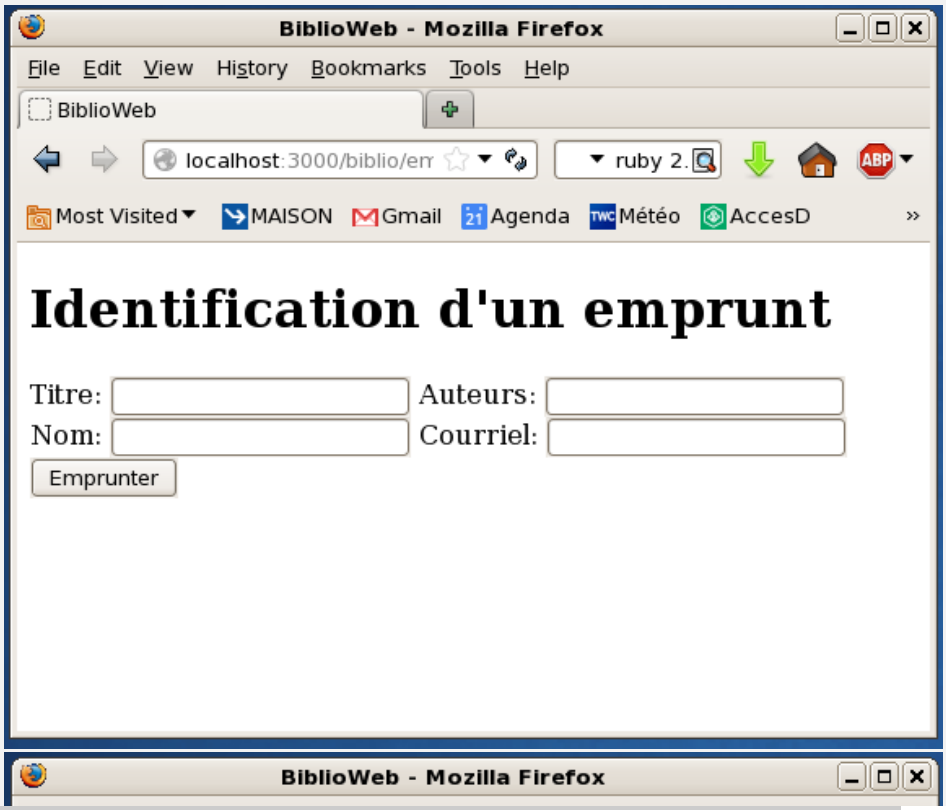
-

`biblio` — version ligne de commandes

```
$ biblio emprunter "Guy T." tremblay.guy@uqam.ca\
        "The RSpec Book" "Chelimsky et al."

$ biblio emprunteur "The RSpec Book"
Guy T.

$ biblio rappeler_livre "The RSpec Book"
Un courriel a ete transmis a tremblay.guy@uqam.ca.

$ biblio rapporter "The RSpec Book"

$ biblio emprunteur "The RSpec Book"
error: Aucun livre emprunte avec le titre
       'The RSpec Book'.
```


- Un exemple qui illustre ces notions, plus particulièrement dans le contexte TDD/BDD
- Pourquoi ligne de commandes ? Parce que c'est ma façon habituelle de travailler. Parce que c'est plus simple à mettre en oeuvre et à expliquer dans un exemple. Parce que c'est donc la première mise en oeuvre que j'ai développée.
- Parce que j'ai aussi déjà une version fonctionnelle en C, développée il y a quelques années par un groupe d'étudiants et qui a ensuite été utilisé comme «corpus de maintenance» dans le cours INF3135.
- Ce qui m'intéresse aujourd'hui, c'est de présenter l'allure générale de la solution, des scénarios et de certains tests, plus spécifiquement pour `rappeler_livre`.

`biblio` — version Web

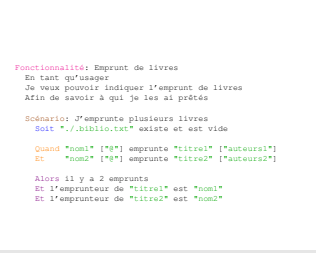


- Mais, plus récemment, j'ai aussi mis en oeuvre une version, simple, avec une interface Web réalisée avec Ruby on Rails.

Spécification `cucumber : emprunter.feature`

(modes ligne de commandes et Web)

```
Fonctionnalité: Emprunt de livres
  En tant qu'usager
  Je veux pouvoir indiquer l'emprunt de livres
  Afin de savoir à qui je les ai prêtés

  Scénario: J'emprunte plusieurs livres
    Soit "./.biblio.txt" existe et est vide

    Quand "nom1" ["@"] emprunte "titre1" ["auteurs1"]
    Et    "nom2" ["@"] emprunte "titre2" ["auteurs2"]

    Alors il y a 2 emprunts
    Et l'emprunteur de "titre1" est "nom1"
    Et l'emprunteur de "titre2" est "nom2"
```

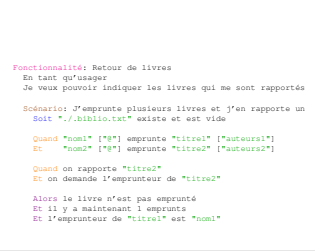



-

Spécification `cucumber` : `rapporter.feature`

(modes ligne de commandes et Web)

```
Fonctionnalité: Retour de livres
  En tant qu'usager
  Je veux pouvoir indiquer les livres qui me sont rapportés

  Scénario: J'emprunte plusieurs livres et j'en rapporte un
    Soit "./.biblio.txt" existe et est vide

    Quand "nom1" ["@"] emprunte "titre1" ["auteurs1"]
    Et    "nom2" ["@"] emprunte "titre2" ["auteurs2"]

    Quand on rapporte "titre2"
    Et on demande l'emprunteur de "titre2"

    Alors le livre n'est pas emprunté
    Et il y a maintenant 1 emprunts
    Et l'emprunteur de "titre1" est "nom1"
```

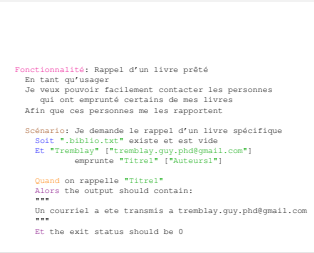



-

Spécification `cucumber : rappeler.feature`

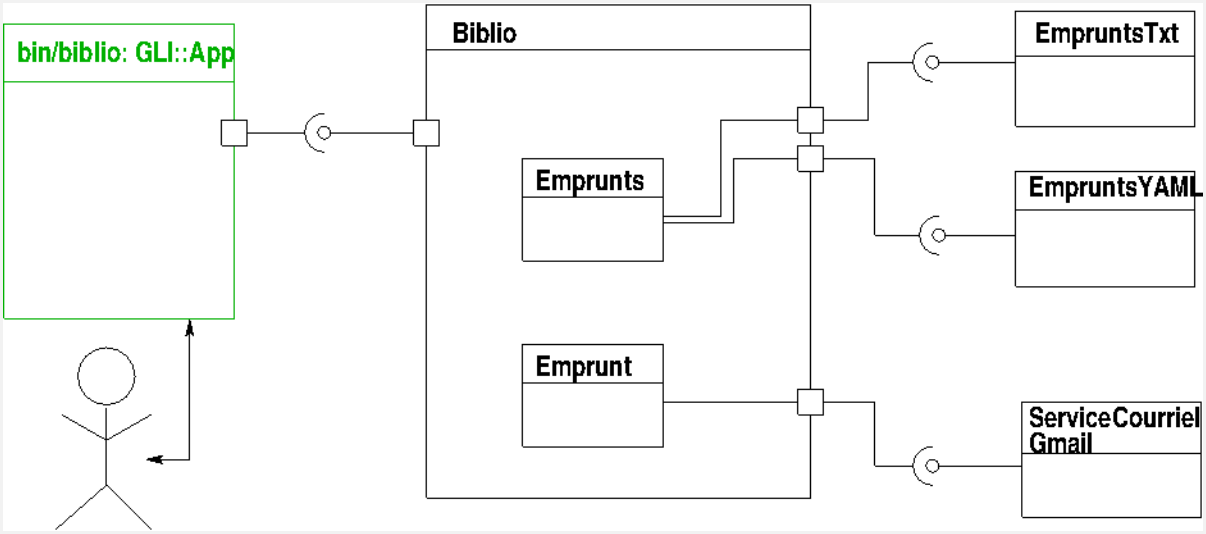
(mode ligne de commandes seulement)

```
Fonctionnalité: Rappel d'un livre prêté
  En tant qu'usager
  Je veux pouvoir facilement contacter les personnes
     qui ont emprunté certains de mes livres
  Afin que ces personnes me les rapportent

  Scénario: Je demande le rappel d'un livre spécifique
    Soit ".biblio.txt" existe et est vide
    Et "Tremblay" ["tremblay.guy.phd@gmail.com"]
            emprunte "Titre1" ["Auteurs1"]

    Quand on rappelle "Titre1"
    Alors the output should contain:
    """
    Un courriel a ete transmis a tremblay.guy.phd@gmail.com
    """
    Et the exit status should be 0
```

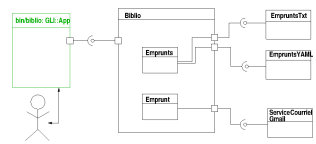



-

Architecture de `biblio`

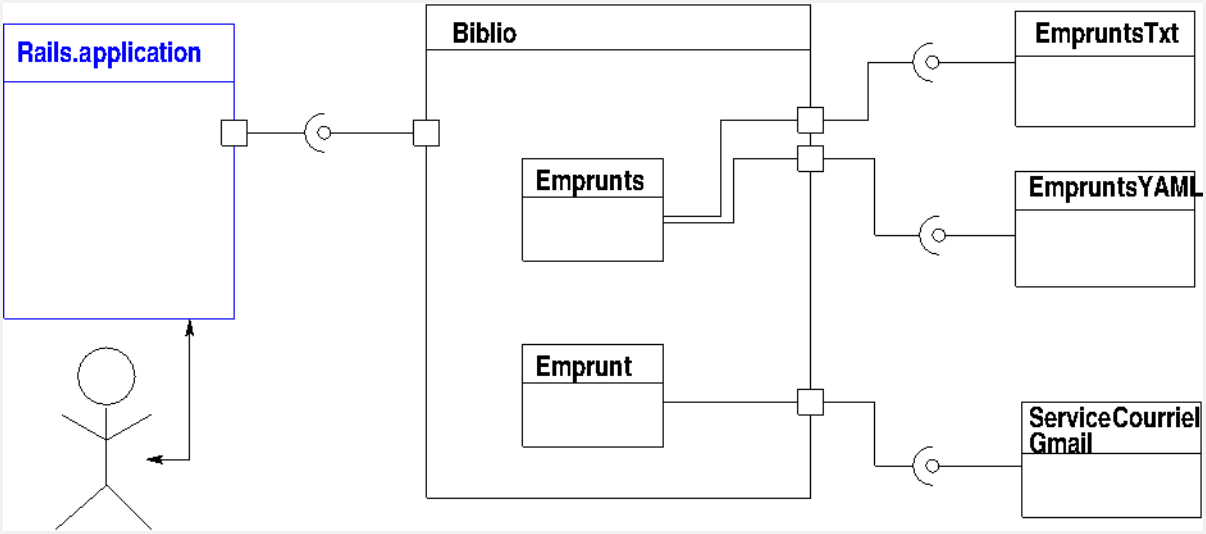


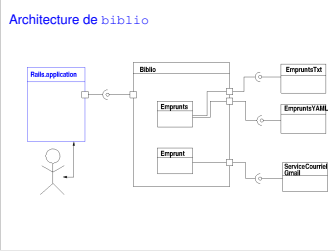
Architecture de `biblio`



- Voici donc ce que ça donne dans le contexte de mon application `biblio` pour la gestion de prêts de livres.

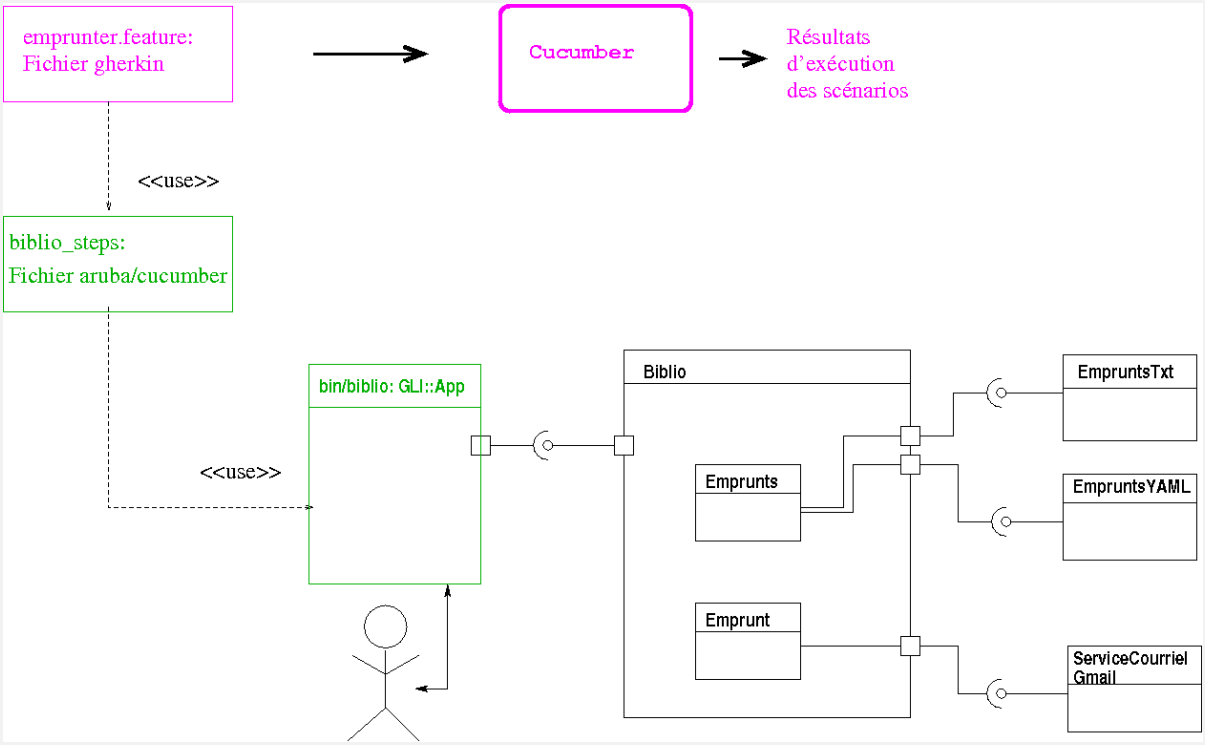
Architecture de `biblio`





• Quand on veut changer d'interface personne-machine pour une application Web, il suffit essentiellement de définir un nouveau composant, qui va utiliser les autres composants déjà définis.

Exécution des scénarios avec Cucumber : Version ligne de commande

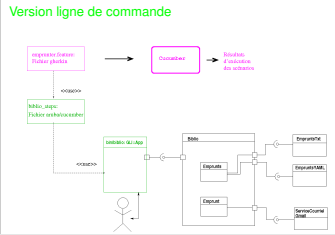


2016-10-14

Tests d'acceptation et BDD(*Behavior Driven Development*)

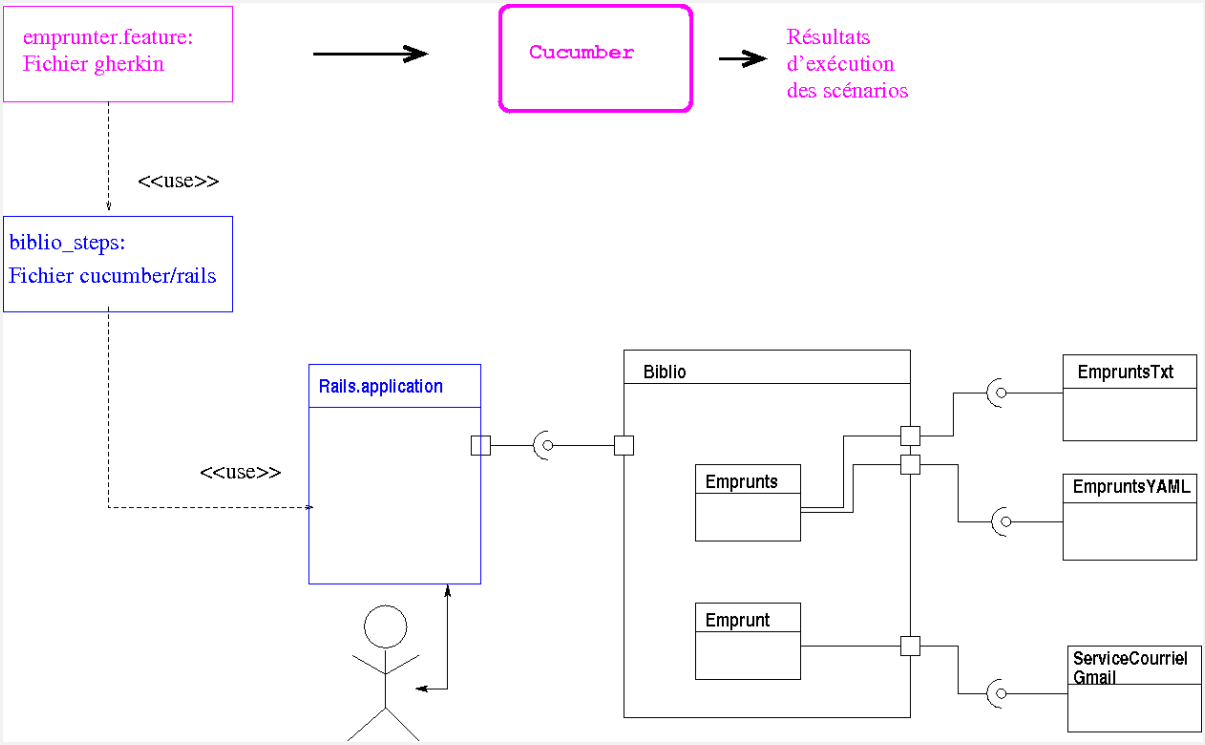
└Un exemple plus détaillé avec `cucumber` : `biblio`

└Spécification de `biblio`



-

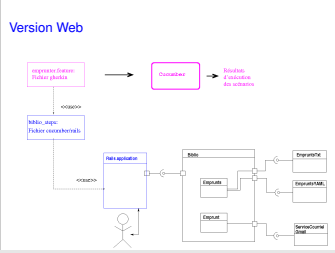
Exécution des scénarios avec Cucumber : Version Web



Tests d'acceptation et BDD(*Behavior Driven Development*)

└Un exemple plus détaillé avec `cucumber` : `biblio`

└Spécification de `biblio`



-

Étapes `cucumber/aruba`, version ligne de commandes : `biblio_steps.rb`

```
Soit(/^"(.*)" existe et est vide$/) do |fich|
  step %{I successfully run
           `bin/biblio --depot=#{fich} init --detruire`}
end


Quand(/^"(.*?)" \["(.*?)"\] emprunte "(.*?)" \["(.*?)"\]$/)
  do |nom, courriel, titre, auteurs|
    step %{I run `bin/biblio emprunter\
                  #{nom} #{courriel} #{titre} #{auteurs}`}




end
```


Étapes `cucumber/aruba`, version ligne de commandes `biblio_steps.rb`

- Le *gem* `aruba` définit un ensemble d'étape `gherkin` prédéfinies qui permettent de définir des conditions, événements, résultats au niveau de l'exécution de commandes au niveau du *shell*.

Étapes `cucumber/rails`, version Web : `biblio_steps.rb`

```
Soit(/^"(.*)" existe et est vide$/) do |fich|
  visit "/biblio/vider"

end


Quand(/^"(.*?)" \["(.*?)"\] emprunte "(.*?)" \["(.*?)"\]$/)
  do |nom, courriel, titre, auteurs|
    visit "/biblio/emprunter"
    fill_in "Titre", :with => titre
    fill_in "Auteurs", :with => auteurs
    fill_in "Nom", :with => nom
    fill_in "Courriel", :with => courriel
    click_button "Emprunter"
end
```


Étapes `cucumber/rails`, version Web : `biblio_steps.rb`

- Mais ces étapes, si elles sont définies à un niveau d'abstractions dans les scénarios, peuvent aussi être mises en oeuvre à l'aide d'opérations sur un fureteur Web.
- Donc, la mise en oeuvre des étapes est spécifique à mon interface personne–machine.

Étapes `cucumber/aruba`, version ligne de commandes : `biblio_steps.rb`

```
Alors(/^l'emprunteur de "(.*?)" est "(.*?)"$/) do |titre, nom|
  step %{I successfully run `bin/biblio emprunteur #{titre}`}
  step %{the stdout should contain "#{nom}"}


end
```


Étapes `cucumber/aruba`, version ligne de commandes : `biblio_steps.rb`

```
Alors(/^l'emprunteur de "(.*?)" est "(.*?)"$/) do |titre, nom|
  step %{I successfully run `bin/biblio emprunteur #{titre}`}
  step %{the stdout should contain "#{nom}"}

end
```

-

Étapes `cucumber/rails`, version Web : `biblio_steps.rb` (suite)

```ruby
Alors(/^l'emprunteur de "(.*?)" est "(.*?)"$/) do |titre, nom|
  visit "/biblio/emprunteur"
  fill_in "Titre recherché", :with => titre
  click_button "Trouver emprunteur"
  expect(page).to have_content(nom)
end
```

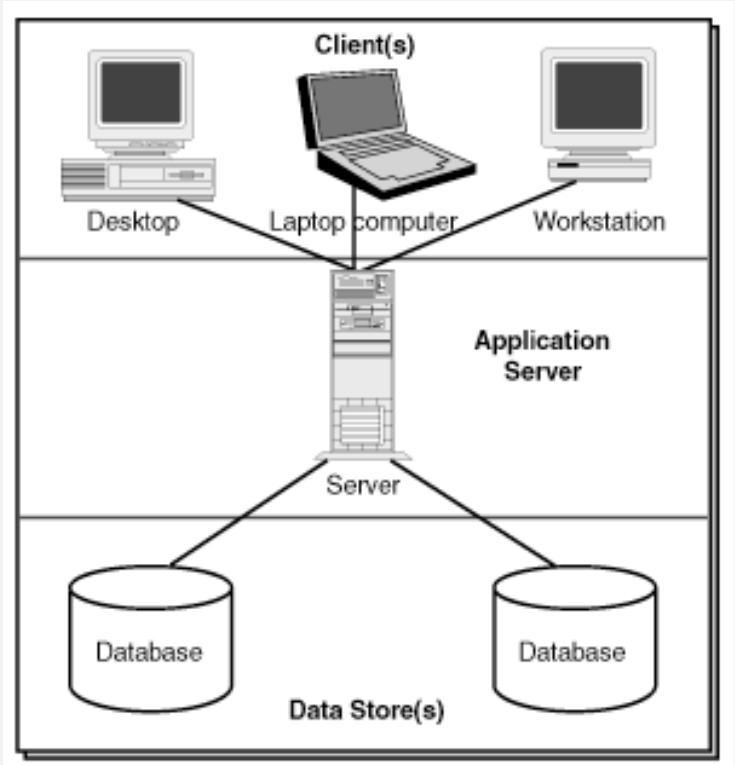

Étapes `cucumber/rails`, version Web : `biblio_steps.rb` (suite)

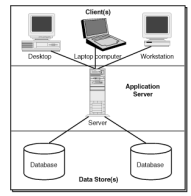
- Donc :
  - On a des scénarios qui sont abstraits
  - On a des étapes qui sont spécifiques/particulières à chacune des interfaces personne–machine.
  - On va voir que ces étapes, liées à chaque IPM, sont mises en oeuvre par des contrôleurs qui vont utiliser la même couche de modèle.

6.2 Mises en oeuvre de `biblio`

-

Architecture en couches : *three tier architecture*

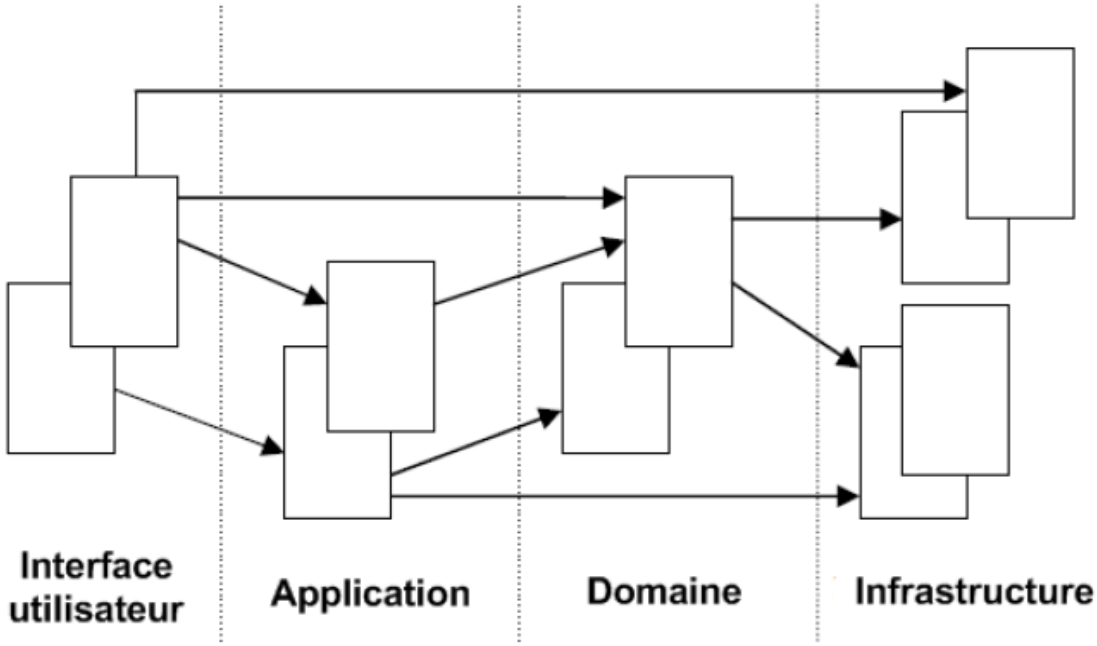


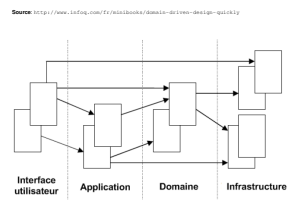


- Tout le monde connait cette architecture en couche, souvent présentée pour illustrer une bonne architecture d'un système fonctionnant avec diverses interfaces persone-machine.

Architecture en couches : *four tier architecture*

Source: `http://www.infoq.com/fr/minibooks/domain-driven-design-quickly`





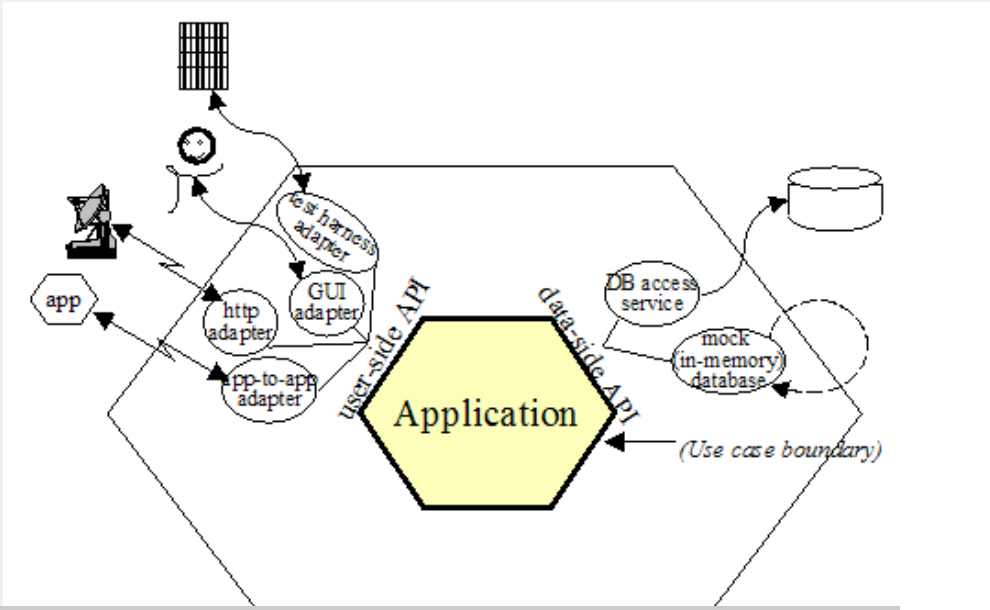
- Une autre variante d'une architecture multi-couche est celle-ci, avec 4 couches, dont une dédiée à la représentation du domaine.

Architecture en couches : *hexagonal (ports and adapters) architecture*

Source: Cockburn, `http://alistair.cockburn.us/Hexagonal+architecture`

Introduite par A. Cockburn, popularisée par «DDD»

Avantage = Tester le modèle (l'application) indépendamment des «services» externes.



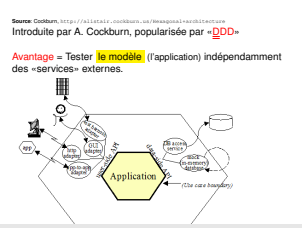
2016-10-14

Tests d'acceptation et BDD(*Behavior Driven Development*)

└Un exemple plus détaillé avec `cucumber` : `biblio`

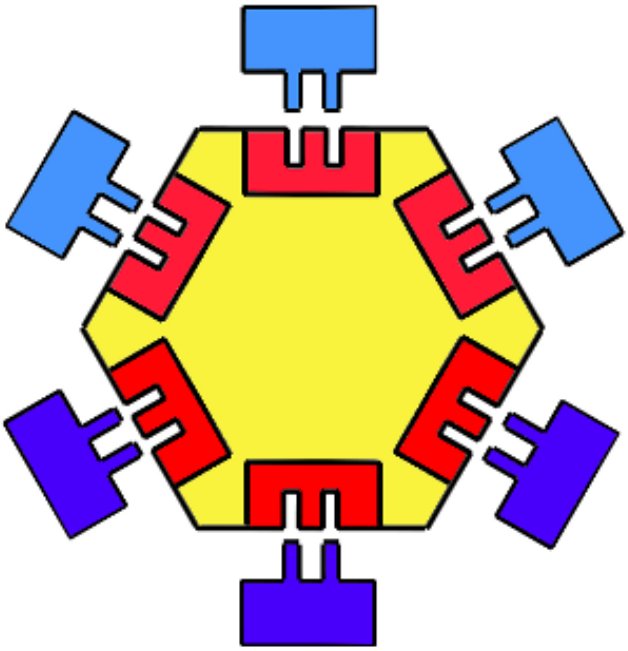
└Mises en oeuvre de `biblio`

└Architecture en couches : *hexagonal (ports and adapters) architecture*

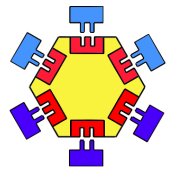


- C'est une forme d'architecture, plus générale que l'approche à quatre couches, qu'on retrouve dans les références plus récentes qui traitent de DDD, par exemple, le bouquin de Vernon, «*Implementing Domain Driven Design*».
- Patron introduit par Alistair Cockburn, circa 2004, initialement sous le nom de «*Hexagonal architecture*», puis sous le nom de «*Ports and adapters architecture*».
- Dixit Cockburn : «Create your application to work without either a UI or a database so you can run automated regression-tests against the application, work when the database becomes unavailable, and link applications together without any user involvement.»
  `http://alistair.cockburn.us/Hexagonal+architecture`
- «Advantages of this architecture : The core logic can be tested independent of outside services. It is easy to replace services by other ones that are more fit in view of changing requirements.»
  `http://www.dossier-andreas.net/software_`

Architecture en couches : *hexagonal (ports and adapters) architecture* (bis)

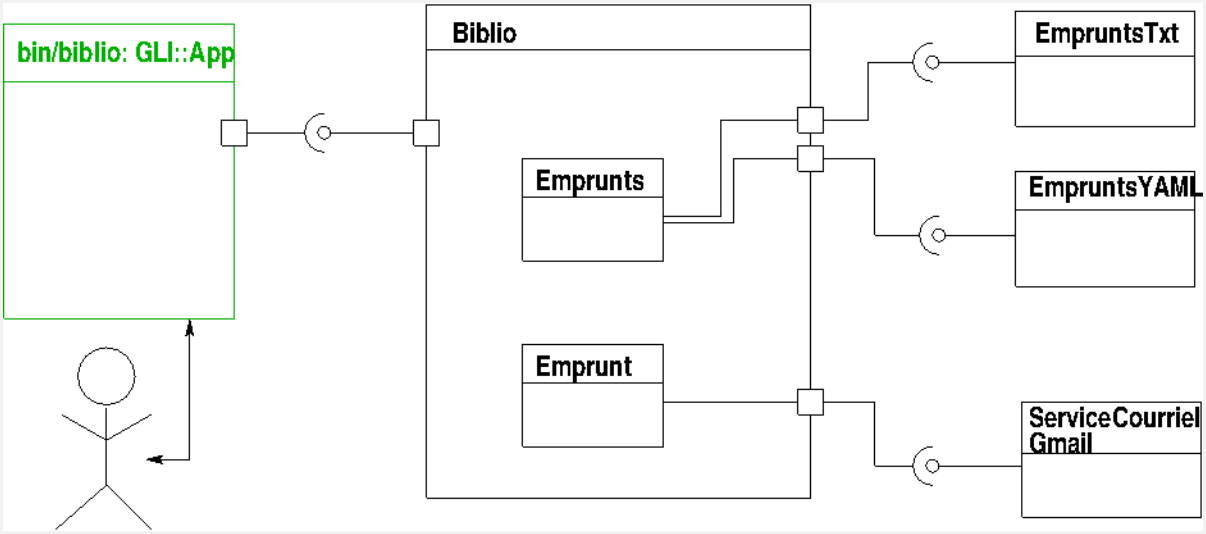


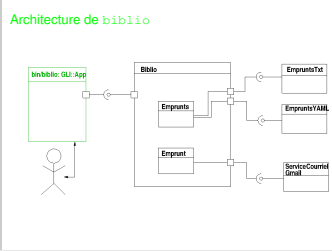
yellow: core logic
light red: primary ports
light blue: primary adapters
dark red: secondary ports
dark blue: secondary adapters



-

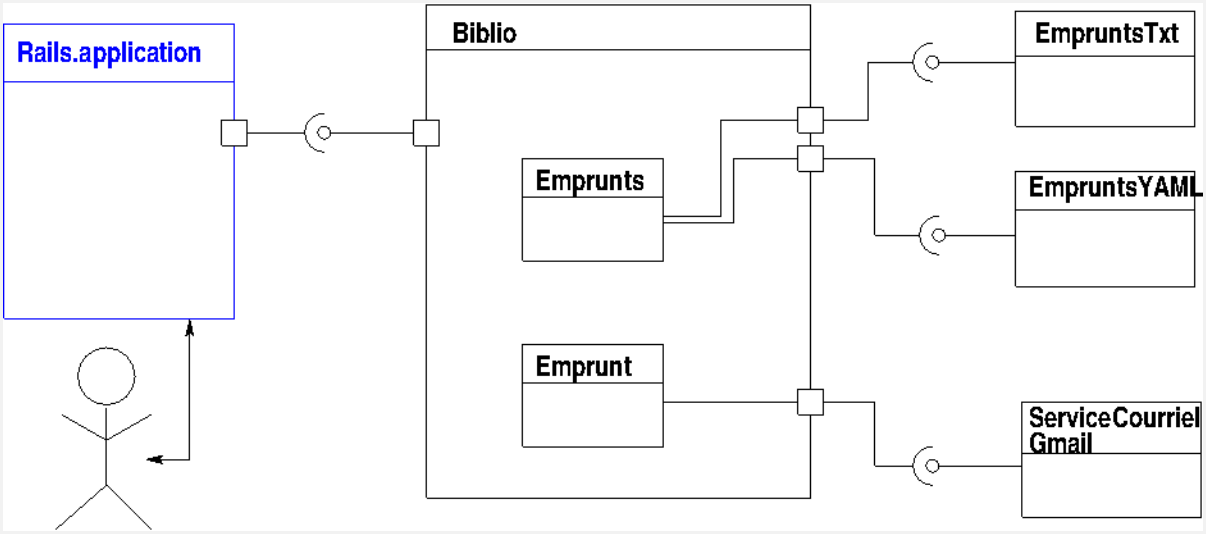
Architecture de `biblio`

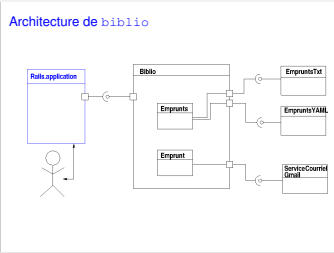




- Voici donc ce que ça donne dans le contexte de mon application `biblio` pour la gestion de prêts de livres.

Architecture de `biblio`

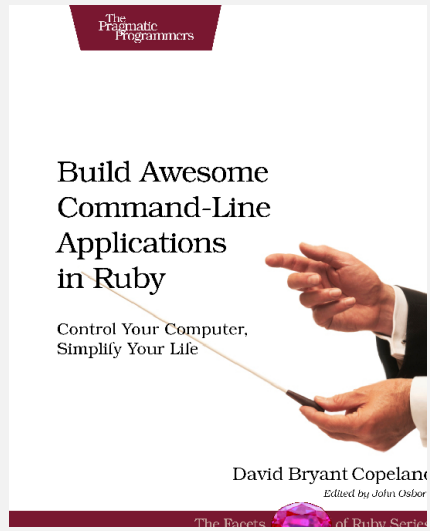


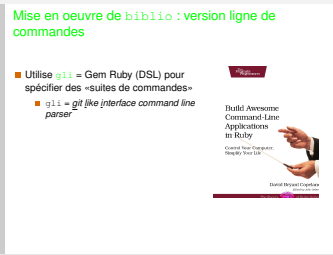


- Quand on veut changer d'interface personne-machine pour une application Web, il suffit essentiellement de définir un nouveau composant, qui va utiliser les autres composants déjà définis.

Mise en oeuvre de `biblio` : version ligne de commandes

- Utilise `gli` = Gem Ruby (DSL) pour spécifier des «suites de commandes»
  - `gli` = *git like interface command line parser*

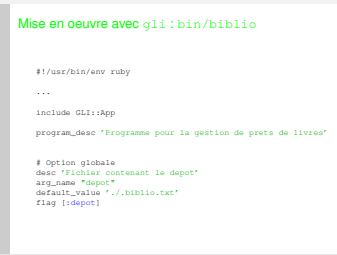




-

Mise en oeuvre avec `gli :bin/biblio`

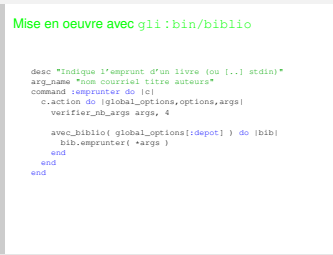
```
#!/usr/bin/env ruby

...

include GLI::App

program_desc 'Programme pour la gestion de prets de livres'


# Option globale
desc 'Fichier contenant le depot'
arg_name "depot"
default_value './.biblio.txt'
flag [:depot]
```

- Présentation descendante
- Le programme principal, la racine de l'exécutable, est `bin/biblio`
- Cet exécutable est un script Ruby, analysé et exécuté par l'interpréteur Ruby grâce au *shebang* = «# !»
- En termes d'architecture hexagonale, le fichier `bin/biblio` représente le «*primary driver*», associé aux interactions avec l'usager

Mise en oeuvre avec `gli :bin/biblio`

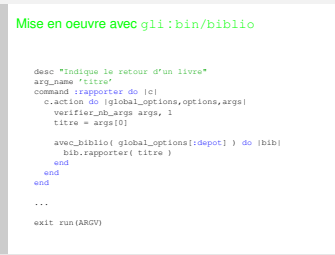
```
desc "Indique l'emprunt d'un livre (ou [..] stdin)"
arg_name "nom courriel titre auteurs"
command :emprunter do |c|
  c.action do |global_options,options,args|
    verifier_nb_args args, 4

    avec_biblio( global_options[:depot] ) do |bib|
      bib.emprunter( *args )
    end
  end
end
```

- Chaque commande de la suite est définie par un appel à `command`, suivi du nom de la commande à définir, suivi d'un bloc qui spécifie les détails de la commande — notamment, le plus important, l'action à exécuter
- Ces différentes commandes représentant, dans la terminologie MVC, les différents controleurs, qui font appel aux opérations du modèle = du domaine.

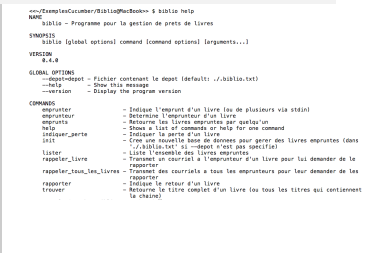
Mise en oeuvre avec `gli :bin/biblio`

```
desc "Indique le retour d'un livre"
arg_name 'titre'
command :rapporter do |c|
  c.action do |global_options,options,args|
    verifier_nb_args args, 1
    titre = args[0]

    avec_biblio( global_options[:depot] ) do |bib|
      bib.rapporter( titre )
    end
  end
end

...

exit run(ARGV)
```

- Une fois les commandes spécifiées, il suffit ensuite simplement d'appeler la méthode `run`, définie dans `GLI::App`.

```
<<~/ExemplesCucumber/Biblio@MacBook>> $ biblio help
NAME
    biblio - Programme pour la gestion de prets de livres

SYNOPSIS
    biblio [global options] command [command options] [arguments...]

VERSION
    0.4.0

GLOBAL OPTIONS
    --depot=depot - Fichier contenant le depot (default: ./.biblio.txt)
    --help        - Show this message
    --version     - Display the program version

COMMANDS
    emprunter                - Indique l'emprunt d'un livre (ou de plusieurs via stdin)
    emprunteur               - Determine l'emprunteur d'un livre
    emprunts                 - Retourne les livres empruntes par quelqu'un
    help                     - Shows a list of commands or help for one command
    indiquer_perte           - Indiquer la perte d'un livre
    init                     - Cree une nouvelle base de donnees pour gerer des livres empruntes (dans
                               './.biblio.txt' si --depot n'est pas specifie)
    lister                   - Liste l'ensemble des livres empruntes
    rappeler_livre           - Transmet un courriel a l'emprunteur d'un livre pour lui demander de le
                               rapporter
    rappeler_tous_les_livres - Transmet des courriels a tous les emprunteurs pour leur demander de les
                               rapporter
    rapporter                - Indique le retour d'un livre
    trouver                  - Retourne le titre complet d'un livre (ou tous les titres qui contiennen
                               la chaine)
```

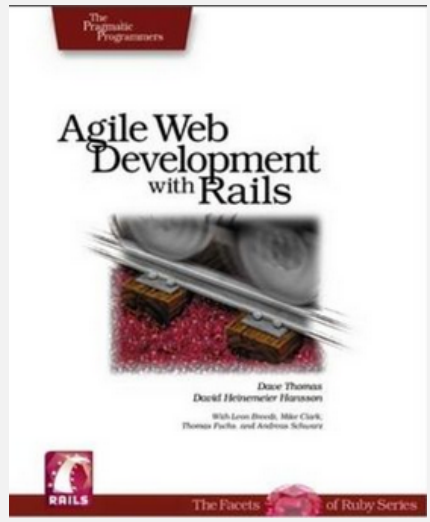



• Par défaut, une commande `help` est automatiquement générée, produisant la documentation illustrée dans la figure ci-haut.

Mise en oeuvre de `biblio` : version Web

- Utilise `rails` = *Framework* Ruby pour développer des applications Web

- Rails utilise une approche de «convention plutôt que configuration»



• Toutefois, faute de temps — ce pourrait être un séminaire complet à lui seul — je ne vous présenterai pas du tout de détails de la mise en oeuvre avec Rails.

Structure du code pour la version Rails (1)

```
|-- app
|   |-- assets
|   |   |-- images
|   |   |-- javascripts
|   |   `-- stylesheets
|   |-- controllers
|   |   |-- application_controller.rb
|   |   |-- biblio_controller.rb
|   |   `-- concerns
|   |-- helpers
    ...
|   |-- mailers
|   |-- models
|   |   `-- concerns
```


-

Structure du code pour la version Rails (2)

```
|   `-- views
|       |-- biblio
|       |   |-- emprunter.html.erb
|       |   |-- emprunteur.html.erb
|       |   |-- fermer.html.erb
|       |   |-- index.html.erb
|       |   |-- l_emprunt.html.erb
|       |   |-- le_titre_a_rapporter.html.erb
|       |   |-- le_titre.html.erb
|       |   |-- lister.html.erb
|       |   |-- rapporter.html.erb
|       |   `-- vider.html.erb
|       `-- layouts
|           `-- application.html.erb
|-- bin
...
```


-

Structure du code pour la version Rails (3)

```
|-- config
...
|   |-- environments
|   |   |-- development.rb
|   |   |-- production.rb
|   |   `-- test.rb
...
|   |-- routes.rb
|   `-- secrets.yml
|-- features
|   |-- emprunter.feature
|   |-- rapporter.feature
|   |-- step_definitions
|   |   `-- biblio_steps.rb
...
|-- lib
...
54 directories, 114 files
```


|-- config
...
| |-- environments
| | |-- development.rb
| | |-- production.rb
| | `-- test.rb
...
| |-- routes.rb
| `-- secrets.yml
|-- features
| |-- emprunter.feature
| |-- rapporter.feature
| |-- step_definitions
| | `-- biblio_steps.rb
...
|-- lib
...
54 directories, 114 files

-

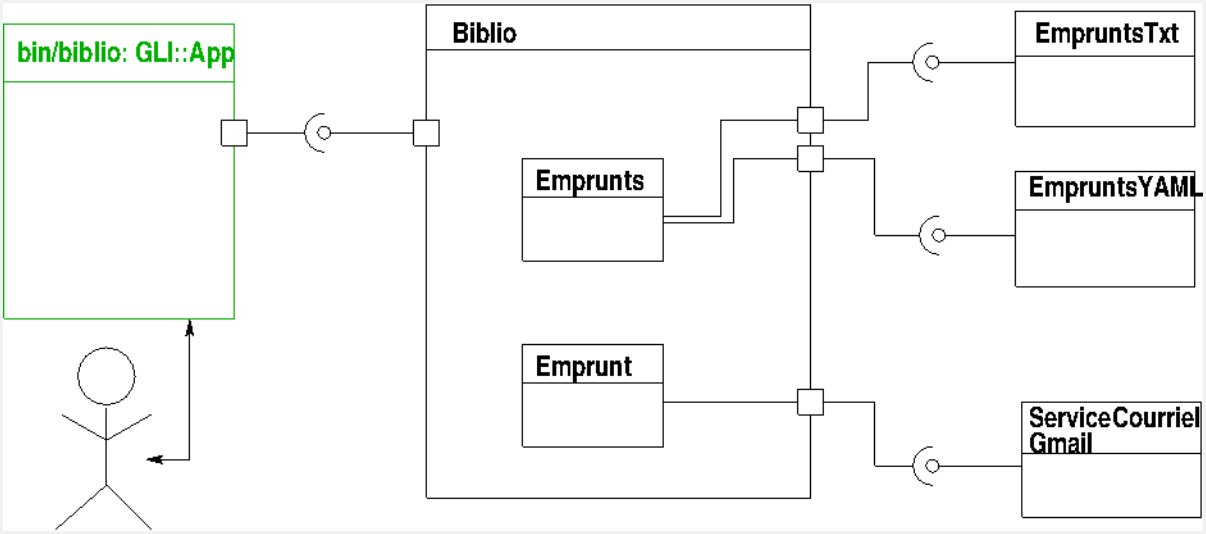
6.3 Tests des services externes (avec RSpec)

-

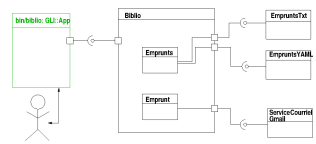
ServiceCourrielGmail.envoyer_courriel

-

Architecture de `biblio`



Architecture de `biblio`



- Voici donc ce que ça donne dans le contexte de mon application `biblio` pour la gestion de prêts de livres.

Mise en oeuvre de `ServiceCourrielGmail.envoyer_courriel`

Fichier `lib/services/service-courriel-gmail.rb`

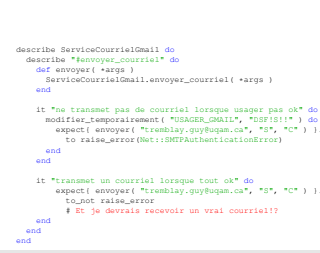
```ruby
module ServiceCourrielGmail

  def self.envoyer_courriel( destinataire, sujet, contenu )
    # Source: http://thinkingeek.com/2012/07/29/sending-emails-google-mail-ruby/
        ...
    Net::SMTP.enable_tls(OpenSSL::SSL::VERIFY_NONE)
    Net::SMTP.start( 'smtp.gmail.com' ... ) do |smtp|
      smtp.send_message( ... )
    end
  end

end
```


- On va maintenant s'attarder aux interactions avec les entités externes, parce que c'est là que cela devient intéressant au niveau des tests.
- Voici la méthode qui permet de transmettre un courriel, en utilisant le compte `gmail` du prêteur de livres. C'est donc cette méthode qui connaît les détails, très techniques, de mise en oeuvre de l'envoi de courriel.
- Et c'est cette méthode qui doit être appelée, directement ou indirectement, par la méthode `rappeler`, qui permet d'envoyer un courriel de rappel à un emprunteur

Tests unitaires de `ServiceCourrielGmail.envoyer_courriel`

Fichier `spec/services/service-courriel_spec.rb`

```ruby
describe ServiceCourrielGmail do
  describe "#envoyer_courriel" do
    def envoyer( *args )
      ServiceCourrielGmail.envoyer_courriel( *args )
    end

    it "ne transmet pas de courriel lorsque usager pas ok" do
      modifier_temporairement( "USAGER_GMAIL", "DSF!S!!" ) do
        expect{ envoyer( "tremblay.guy@uqam.ca", "S", "C" ) }.
          to raise_error(Net::SMTPAuthenticationError)
      end
    end

    it "transmet un courriel lorsque tout ok" do
        expect{ envoyer( "tremblay.guy@uqam.ca", "S", "C" ) }.
          to_not raise_error
          # Et je devrais recevoir un vrai courriel!?
    end
  end
end
```

- Ce qu'il est intéressant de regarder, c'est l'intérêt que cette approche a sur la forme des tests.
- Voici donc, dans un premier temps, les tests unitaires, exprimés en RSpec, pour `ServiceCourrielGmail`, donc pour le «vrai» service : on vérifie différents cas d'erreur et on vérifie «de façon non automatique» pour ce cas particulier, que l'envoi de courriel s'effectue correctement.

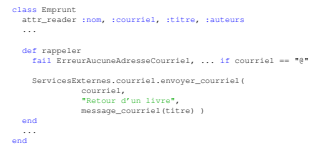
Emprunt#rappeler

-

Mise en oeuvre de `Emprunt#rappeler`

Fichier `lib/biblio/emprunt.rb`

```
class Emprunt
  attr_reader :nom, :courriel, :titre, :auteurs
  ...

  def rappeler
    fail ErreurAucuneAdresseCourriel, ... if courriel == "@"

    ServicesExternes.courriel.envoyer_courriel(
              courriel,
              "Retour d'un livre",
              message_courriel(titre) )
  end
  ...
end
```

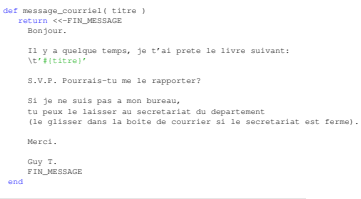



-

Mise en oeuvre de `Emprunt#rappeler` (suite)

Fichier `lib/biblio/emprunt.rb`

```
def message_courriel( titre )
   return <<-FIN_MESSAGE
     Bonjour.

     Il y a quelque temps, je t'ai prete le livre suivant:
     \t'#{titre}'

     S.V.P. Pourrais-tu me le rapporter?

     Si je ne suis pas a mon bureau,
     tu peux le laisser au secretariat du departement
     (le glisser dans la boite de courrier si le secretariat est f

     Merci.

     Guy T.
     FIN_MESSAGE
 end
```



def message_courriel(titre)
return <<-FIN_MESSAGE
Bonjour.

Il y a quelque temps, je t'ai prete le livre suivant:
\t'#{titre}'

S.V.P. Pourrais-tu me le rapporter?

Si je ne suis pas a mon bureau,
tu peux le laisser au secretariat du departement
(le glisser dans la boite de courrier si le secretariat est ferme).

Merci.

Guy T.
FIN_MESSAGE
end

-

Mise en oeuvre de `Emprunt#rappeler` et injection de dépendances par un registre de services

Dans le fichier `bin/biblio` :

```
Biblio::ServicesExternes.courriel = ServiceCourrielGmail
```

Dans le fichier `lib/biblio/emprunt.rb` :

```
class Emprunt
  attr_reader :nom, :courriel, :titre, :auteurs
  ...

  def rappeler
    ...
    ServicesExternes.courriel.envoyer_courriel(
              courriel,
              "Retour d'un livre",
              message_courriel(titre) )
  end
  ...
end
```


- Dans le programme principal (`bin/biblio`), on définit une variable globale qui identifie quel service d'envoi de courriel doit être utilisé.
- Dans la méthode `rappeler`, on réfère à cette variable pour obtenir le nom du service à utiliser, objet/module sur lequel appelle alors la méthode `envoyer_courriel`.

Mise en oeuvre de `Emprunt#rappeler` et injection de dépendances par un registre de services

Dans le fichier `bin/biblio` :

```
Biblio::ServicesExternes.courriel = ServiceCourrielGmail
```

Dans le fichier `lib/biblio/emprunt.rb` :

```
class Emprunt
  attr_reader :nom, :courriel, :titre, :auteurs
  ...

  def rappeler
    ...
    ServicesExternes.courriel.envoyer_courriel(
              courriel,
              "Retour d'un livre",
              message_courriel(titre) )
  end
  ...
end
```

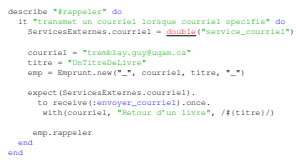
⇒ Respecte le DIP ☺

- C'est une forme «d'injection de dépendances» : style *setter injection*.
- Cela peut aussi être vu comme une forme de *service locator* = «*This pattern uses a central registry known as the "service locator", which on request returns the information necessary to perform a certain task*» : `http://en.wikipedia.org/wiki/Service_locator_pattern`
- Autre nom : La variable globale joue le rôle d'un registre des services (*service registry*).

Tests unitaires de `Emprunt#rappeler`

Fichier `spec/biblio/emprunt_spec.rb`

```ruby
describe "#rappeler" do
  it "transmet un courriel lorsque courriel specifie" do
    ServicesExternes.courriel = double("service_courriel")

    courriel = "tremblay.guy@uqam.ca"
    titre = "UnTitreDeLivre"
    emp = Emprunt.new("_", courriel, titre, "_")

    expect(ServicesExternes.courriel).
      to receive(:envoyer_courriel).once.
       with(courriel, "Retour d'un livre", /#{titre}/)

     emp.rappeler
  end
end
```

- Voici maintenant les tests pour la méthode `rappeler` de la classe `Emprunt`, qui utilise le service d'envoi de courriel.

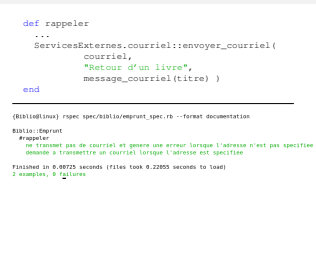
On n'a pas besoin de tester à nouveau l'envoi effectif de courriel, car cela a déjà été fait dans les tests pour `ServiceCourrielGmail`.

Et ici, dans ce test, on ne veut pas non plus dépendre spécifiquement du service de gmail, car rien nous dit que c'est ce service qui est utilisé, i.e., un tout autre service pourrait très bien être utilisé à la place. Ce n'est pas à `rappeler` à savoir cela.

Ici, on a simplement besoin de s'assurer que le service, quel qu'il soit, reçoive la demande appropriée. C'est ce qu'on fait à l'aide d'un «*test double*» et à l'aide d'attentes explicites (*expectations*) sur le comportement observé.

```ruby
def rappeler
  ...
  ServicesExternes.courriel::envoyer_courriel(
            courriel,
            "Retour d'un livre",
            message_courriel(titre) )
end
```

```
{Biblio@linux} rspec spec/biblio/emprunt_spec.rb --format documentation

Biblio::Emprunt
  #rappeler
    ne transmet pas de courriel et genere une erreur lorsque l'adresse n'est pas specifiee
    demande a transmettre un courriel lorsque l'adresse est specifiee

Finished in 0.00725 seconds (files took 0.22055 seconds to load)
2 examples, 0 failures
```

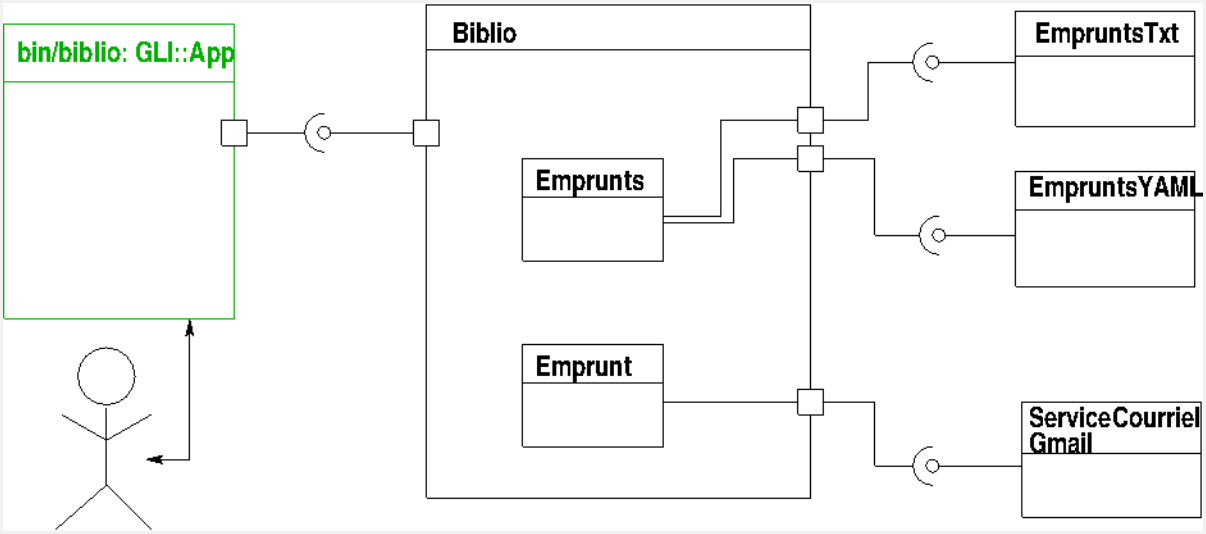



- Ici, on voit ce qui est affiché/indiqué si l'*expectation* est satisfaite — en format `documentation`.

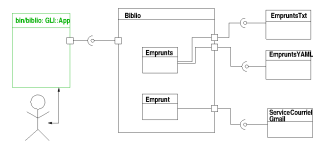
EmpruntsTxt#charger

-

Architecture de `biblio`



Architecture de `biblio`

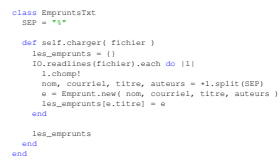


- Voici donc ce que ça donne dans le contexte de mon application `biblio` pour la gestion de prêts de livres.

Mise en oeuvre de `EmpruntsTxt#charger`

Fichier `lib/biblio/emprunts-txt.rb`

```
class EmpruntsTxt
  SEP = "%"

  def self.charger( fichier )
    les_emprunts = {}
    IO.readlines(fichier).each do |l|
      l.chomp!
      nom, courriel, titre, auteurs = *l.split(SEP)
      e = Emprunt.new( nom, courriel, titre, auteurs )
      les_emprunts[e.titre] = e
    end

    les_emprunts
  end
end
```

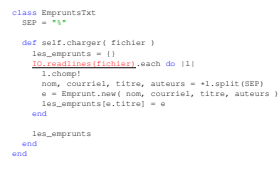



- Voici un autre exemple, cette fois pour le service externe qui charge en mémoire le contenu de la base de données, lorsque celle-ci est en format textuelle.

Mise en oeuvre de `EmpruntsTxt#charger`

Fichier `lib/biblio/emprunts-txt.rb`

```
class EmpruntsTxt
  SEP = "%"

  def self.charger( fichier )
    les_emprunts = {}
    IO.readlines(fichier).each do |l|
      l.chomp!
      nom, courriel, titre, auteurs = *l.split(SEP)
      e = Emprunt.new( nom, courriel, titre, auteurs )
      les_emprunts[e.titre] = e
    end

    les_emprunts
  end
end
```

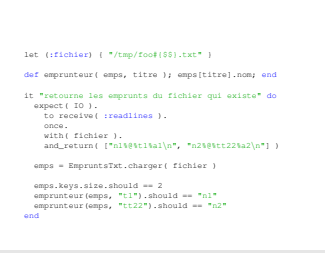



- Les détails ne sont pas importants. L'aspect important sur lequel j'insiste est que la méthode `IO.readlines` est utilisée pour lire le contenu du fichier contenant cette base de données, méthode qui doit donc faire un accès externe à un fichier.
- Or, en autant que c'est possible, notamment pour des raisons de performance, il est préférable de limiter les accès à des fichiers externes dans les tests.

Tests unitaires de `EmpruntsTxt#charger`

Fichier `spec/biblio/emprunts-txt_spec.rb`

```ruby
let (:fichier) { "/tmp/foo#{$$}.txt" }

def emprunteur( emps, titre ); emps[titre].nom; end

it "retourne les emprunts du fichier qui existe" do
  expect( IO ).
    to receive( :readlines ).
    once.
    with( fichier ).
    and_return( ["n1%@%t1%a1\n", "n2%@%tt22%a2\n"] )

  emps = EmpruntsTxt.charger( fichier )

  emps.keys.size.should == 2
  emprunteur(emps, "t1").should == "n1"
  emprunteur(emps, "tt22").should == "n2"
end
```

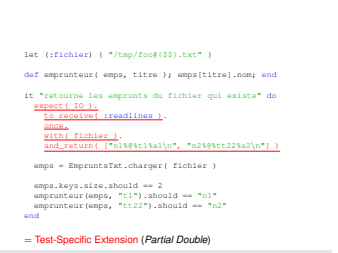
- Voici donc une façon de définir un test pour `charger` qui fait en sorte de ne pas avoir d'accès à un fichier externe, et ce en utilisant un «*partial double* », ce qu'on appelle aussi «*a tests-specific extension*» = «*an extension of a real object in a system that is instrumented with test-double like behaviour in the context of a test*» : `https://github.com/rspec/rspec-mocks`.
- Donc, on dit à l'objet `IO` que temporairement, le temps du test, il doit modifier son comportement pour la méthode `readlines` de façon à ce qu'elle retourne le tableau indiqué si elle reçoit les argument indiqués. Sinon, une erreur doit être signalée si cette méthode n'est pas appelée avec ces arguments.

Tests unitaires de `EmpruntsTxt#charger`

Fichier `spec/biblio/emprunts-txt_spec.rb`

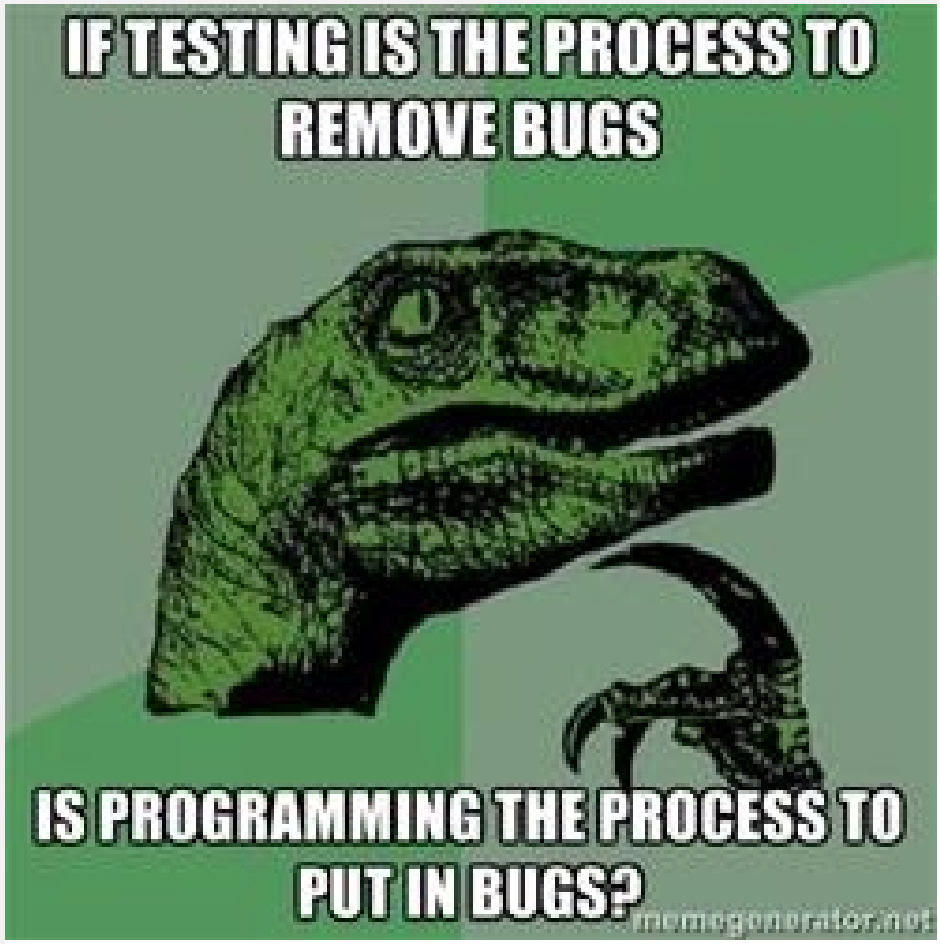
```ruby
let (:fichier) { "/tmp/foo#{$$}.txt" }

def emprunteur( emps, titre ); emps[titre].nom; end

it "retourne les emprunts du fichier qui existe" do
  expect( IO ).
    to receive( :readlines ).
    once.
    with( fichier ).
    and_return( ["n1%@%t1%a1\n", "n2%@%tt22%a2\n"] )

  emps = EmpruntsTxt.charger( fichier )

  emps.keys.size.should == 2
  emprunteur(emps, "t1").should == "n1"
  emprunteur(emps, "tt22").should == "n2"
end
```

= Test-Specific Extension (*Partial Double*)



- Une telle extension, temporaire et spécifique au test, de `IO.readlines` est beaucoup plus simple à définir… que si on avait dû créer un fichier externe `foo.txt` et y mettre comme contenu les lignes désirées.
- En fait, noobstant l'aspect performance, cela permet aussi une meilleure localité des informations pour ce test — pour que le contenu soit clairement local au test, il aurait fallu créer ce fichier au moment du test (ouverture en création/écriture), à partir du contenu désiré, et ensuite lire son contenu (ouverture, implicite, en lecture avec `readlines`).

7. Conclusion


IF TESTING IS THE PROCESS TO REMOVE BUGS
IS PROGRAMMING THE PROCESS TO PUT IN BUGS?
memegenerator.net

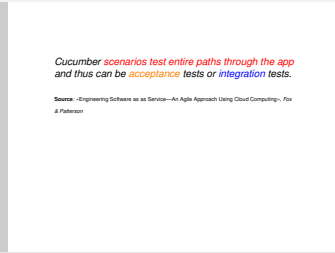

IF TESTING IS THE PROCESS TO REMOVE BUGS
IS PROGRAMMING THE PROCESS TO PUT IN BUGS?

-

Les tests d'acceptation servent aussi de tests d'intégration

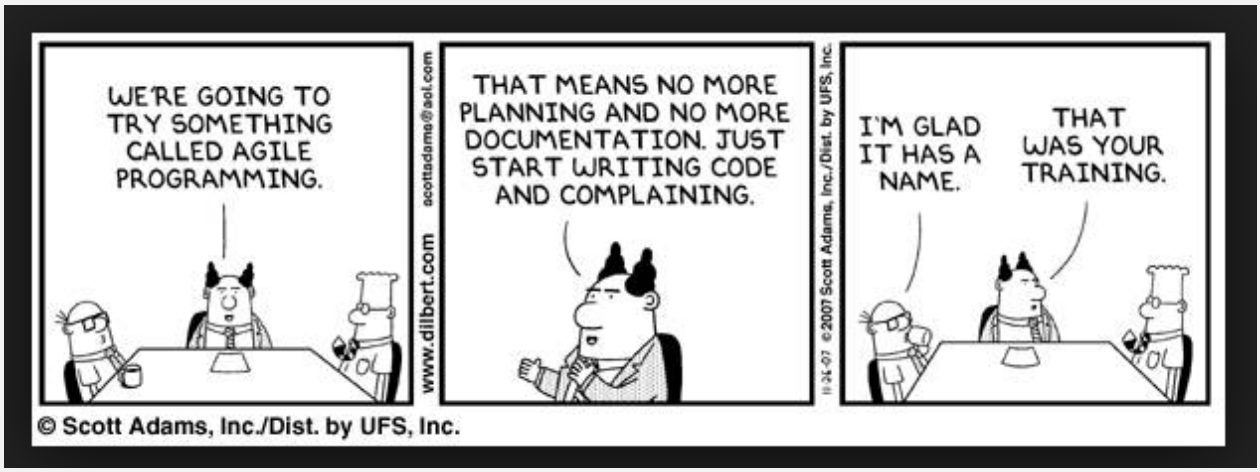
Cucumber scenarios test entire paths through the app and thus can be acceptance tests or integration tests.

Source*:* «Engineering Software as as Service—An Agile Approach Using Cloud Computing», *Fox & Patterson*



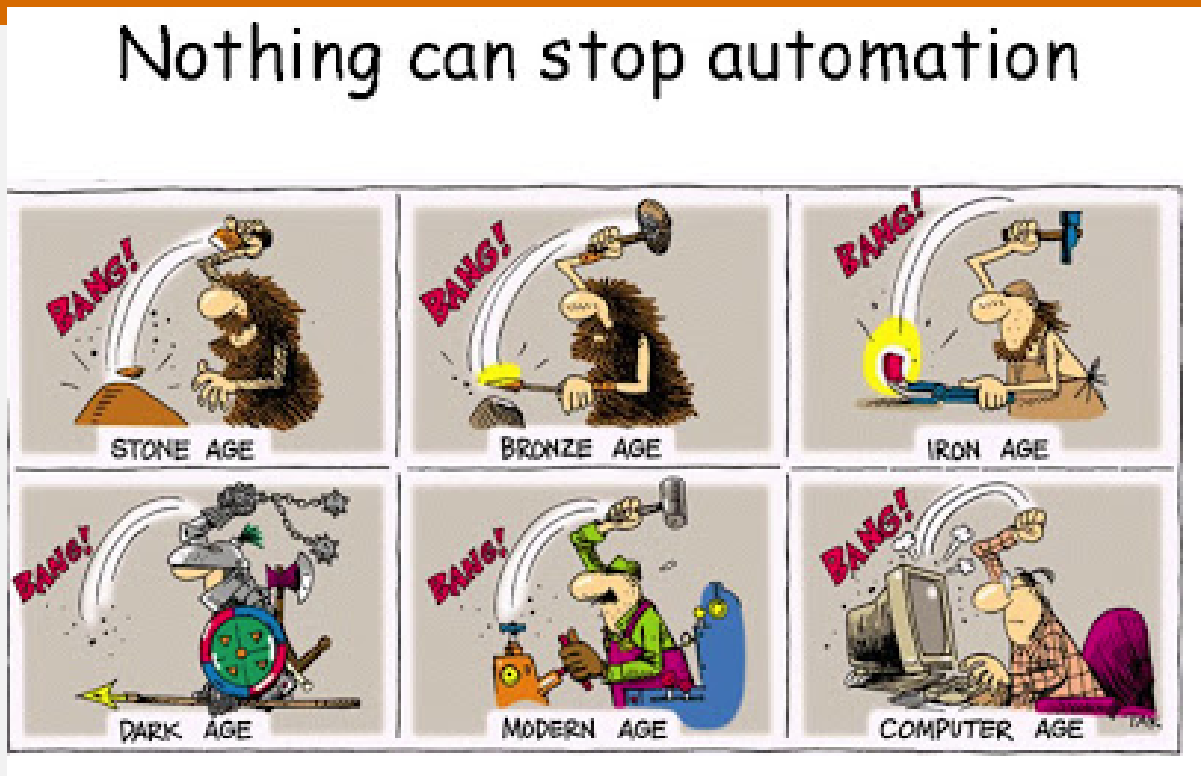
-

Ce sont les méthodes approches agiles ont introduit et popularisé les approches TDD et BDD



-

On peut automatiser les tests unitaires, les tests d'acceptation, etc.



Nothing can stop automation

-

Références

D. Chelimsky, D. Astels, Z. Dennis, A. Hellesoy, B. Helmkamp, and D. North.
The RSpec Book : Behaviour Driven Development with RSpec, Cucumber, and Friends.
The Pragmatic Bookshelf, 2010.

D.B. Copeland.
Build Awesome Command-Line Applications in Ruby : Control Your Computer, Simplify Your Life.
The Pragmatic Bookshelf, 2012.

E. Evans.
Domain-Driven Design—Tackling Complexity in the Heart of Software.
Addison-Wesley, 2004.

A. Fox and D. Patterson.
Engineering Software as as Service—An Agile Approach Using Cloud Computing.
Strawberry Canyon LLC, 2013.

R. Mugridge and W. Cunningham.
Fit for Developing Software—Framework for Integrated Tests.
Prentice-Hall, 2005.

J.F. Smart.
BDD In Action.
Manning, 2015.

M. Wynne and A. Hellesoy.
The Cucumber Book : Behaviour-Driven Development for Testers and Developers.
The Pragmatic Bookshelf, 2012.