

Comment bien rater une
approche

Behavior Driven Development

Matthias Meulien - Sud-Ouest Days - 17 mars 2023

Quésaco ?

Behavior Driven Development

Suites logiques

2 3 5 7 11 13 17 19 23 29 31 37 41 43 ? 47

Observation : on peut essayer de s'appuyer sur des **exemples concrets** pour formaliser un comportement

Quésaco ?

Behavior Driven Development

Introducing BDD, Dan North, *Better Software Magazine Article*, 2006

En s'appuyant sur des exemples concrets, un language polyvalent pourrait permettre de

- Décrire le comportement attendu d'un système
- Exprimer les critères d'acceptation d'une implémentation

Retombées attendues

- Faciliter la collaboration
- Constituer une documentation
- Automatisation de la vérification des critères d'acceptation

Mise en œuvre & outillage

Language

Gherkin

```
# Example adapted from https://cucumber.io
@tag
Feature: Eating too many cucumbers may not be good for you
```

Eating too much of anything may not be good for you

```
Scenario: Eating too much is a problem
Given Alice is hungry
When she eats 4 cucumbers
Then she should be "sick"
```

```
Scenario Outline: Eating a few is no problem
Given Alice is hungry
And Alice feels good
When she eats <number> cucumbers
Then she should feel <feeling>
```

number	feeling
2	good
3	full

Mise en œuvre & outillage

Quadriels

- Cucumber

Basée sur Gherkin, automatise la vérification des critères d'acceptation, multi-language, bien plus qu'une bibliothèque : un écosystème (extension pour Jira, éditeur, etc.)

- behave

Basée sur Gherkin, automatise la vérification des critères d'acceptation, Python, pas hyper dynamique mais très bien documentée et stable

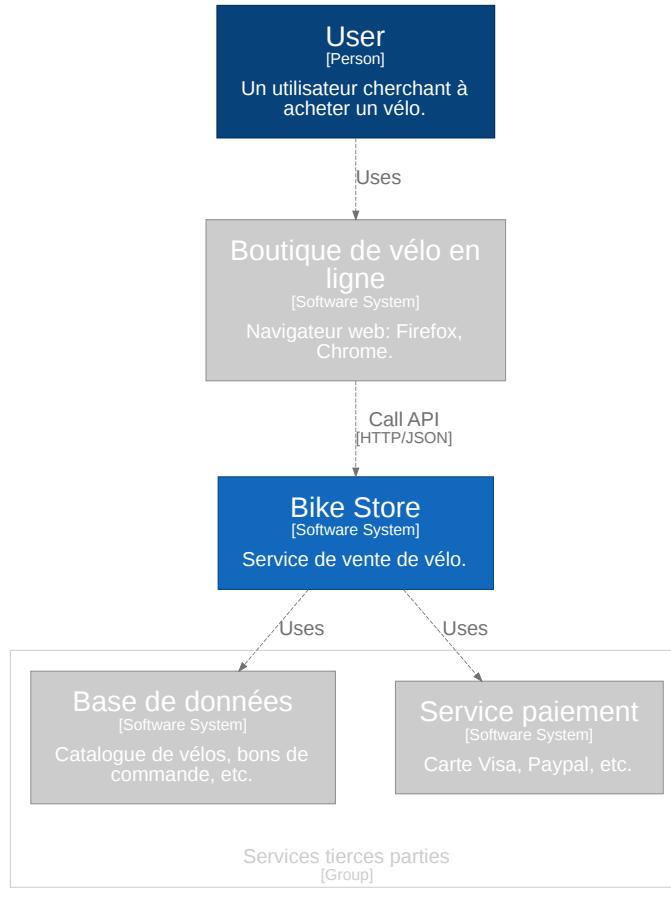
- RSpec

Orientée Ruby, bibliothèque historique, implémentation des critères d'acceptation concomittante avec leur description

Bike store

Contexte

L'API HTTP d'une boutique en ligne de vente de vélo



Bike store

Exemple de Gherkin

Fonctionnalité: Recherche de vélo par marque

Scénario: Cas d'une marque en stock

Étant donné qu'un vélo de marque "Motobécane" est en stock

Quand un utilisateur recherche les vélos de marque "Motobécane"

Alors la liste récupérée devrait "contenir le vélo"

Scénario: Cas d'une marque hors stock

Étant donné que le stock ne contient pas de vélo de marque "Triban"

Quand un utilisateur recherche les vélos de marque "Triban"

Alors la liste récupérée devrait "être vide"

Bike store

Exemple d'automatisation des critères d'acceptation

Scénario: Cas d'une marque en stock

Étant donné qu'un vélo de marque "Motobécane" est en stock

Quand un utilisateur recherche les vélos de marque "Motobécane"

Alors la liste récupérée devrait "contenir le vélo"

```
@given("qu'un vélo de marque {brand} est en stock")
def step(ctx: Context, brand: str) → None:
    bike_desc = ctx.docs[brand]
    ctx.current_bike_doc = ctx.db_client.replace_one(bike_desc, upsert=True)
```

```
@when("un utilisateur recherche les vélos de marque {brand}")
def step(ctx: Context, brand: str) → None:
    url = "/bikes/find-by-brand"
    ctx.resp = ctx.http_client.get(url, params={"brand": brand})
```

```
@then("la liste récupérée devrait {expected_state}")
def step(ctx: Context, expected_state: str) → None:
    assert("resp" in ctx and ctx.resp.status = 200)
    if expected_state = "être vide":
        assert(len(ctx.resp.data) = 0)
    elif expected_state = "contenir le vélo":
        assert("current_bike_doc" in ctx and ctx.current_bike_doc.id in [d.id for d in ctx.resp.data])
```

Bike store

Interlude : le PO enrichit la spécification

Fonctionnalité: Recherche de vélo par marque

Scénario: Cas d'une marque en stock

Étant donné qu'un vélo de marque "Motobécane" est en stock

Quand un utilisateur recherche les vélos de marque "Motobécane"

Alors la liste récupérée devrait " contenir le vélo "

Scénario: Cas d'une marque hors stock

Étant donné que le stock ne contient pas de vélo de marque "Triban"

Et qu'un vélo de marque "Motobécane" est en stock

Quand un utilisateur recherche les vélos de marque "Triban"

Alors la liste récupérée devrait " être vide "

Bike store

Interlude : le PO enrichit la spécification

Fonctionnalité: Recherche de vélo par marque

Scénario: Cas d'une marque en stock

Étant donné qu'un vélo de marque "Motobécane" est en stock

Quand un utilisateur recherche les vélos de marque "Motobécane"

Alors la liste récupérée devrait "contenir le vélo"

Scénario: Cas d'une marque hors stock

Étant donné que le stock ne contient pas de vélo de marque "Triban"

Quand un utilisateur recherche les vélos de marque "Triban"

Alors la liste récupérée devrait "être vide"

Scénario: La recherche est sensible à la casse

Étant donné qu'un vélo de marque "Motobécane" est en stock

Quand un utilisateur recherche les vélos de marque "motobécane"

Alors la liste récupérée devrait "être vide"

Retour sur Mise en œuvre & outillage

Quadriels

RSpec est documenté (mais pas testé) en Gherkin !

Feature: `--tag` option

Use the `--tag` (or `-t`) option to run examples that match a specified tag.
The tag can be a simple `name` or a `name:value` pair.

If a simple `name` is supplied, only examples with `:name => true` will run.
If a `name:value` pair is given, examples with `name => value` will run,
where `value` is always a string. In both cases, `name` is converted to a symbol.

Tags can also be used to exclude examples by adding a `~` before the tag. For example, `~tag` will exclude all examples marked with `:tag => true` and `~tag:value` will exclude all examples marked with `:tag => value`.

Filtering by tag uses a hash internally, which means that you can't specify multiple filters for the same key. For instance, if you try to exclude `:name => 'foo'` and `:name => 'bar'`, you will only end up excluding `:name => 'bar'`.

To be compatible with the Cucumber syntax, tags can optionally start with an

Ecueils à éviter

- Se tromper de langage ou exposer des choix d'implémentation

Fonctionnalité: Recherche de vélo par marque

Scénario: Cas d'une marque en stock

Étant donné que la base de données contient le document

"""

```
{"id": 36, "brand": "Motobécane",
 "color": "red", "size": 60, "status": "sold",
 "acquisition_date": "2019-12-15T22:33:01Z"}
```

"""

Quand on requête l'URL `"/bikes/find-by-brand"` avec la méthode `"GET"` et la query string `?brand=Motobécane"`

Alors le statut de la réponse est `"200"`

Et le JSON reçu est une liste qui contient un élément avec l'identifiant `"36"`

👉 Collaboration impossible, perte de l'aspect documentation

- S'imposer de ne mettre qu'un scénario par fonctionnalité

👉 Perte de l'aspect documentation

- S'imposer d'aller systématiquement jusqu'à l'automatisation

👉 Perte de l'aspect documentation

Ecueils à éviter

- Manquer de rigueur dans la formulation des critères d'acceptation

Fonctionnalité: Recherche de vélo par marque

Scénario: Cas d'une marque en stock

Étant donné qu'un vélo de marque "Motobécane" est en stock

Quand un utilisateur recherche les vélos de marque "Motobécane"

Alors la liste récupérée devrait "contenir le vélo"

Scénario: Cas d'une marque hors stock

Étant donné que le stock ne contient pas de vélo de marque "Triban"

Quand l'utilisateur requête les vélos "Triban"

Alors la liste récupérée devrait "être vide"

👉 Impossible de réutiliser les implémentations des étapes entre scénarios

- Écrire des Gherkin et des critères d'acceptation

👉 Travail en double ?

Ecueils à éviter

- Mal anticiper le niveau de généricité utile à la réutilisation

Fonctionnalité: Recherche de vélo par marque

Scénario: Cas d'une marque en stock

Étant donné qu'un vélo de marque Motobécane est en stock

Quand un utilisateur recherche les vélos de marque Motobécane

Alors le vélo Motobécane devrait être trouvé

Scénario: Cas d'une marque hors stock

Étant donné que le stock ne contient aucun vélo de marque Triban

Quand un utilisateur recherche les vélos de marque Triban

Alors aucun vélo ne sera trouvé

👉 Impossible de réutiliser les implémentations des étapes entre scénarios

- Bouchonner plutôt que peupler une base de données

👉 Fragile !

Ecueils à éviter

- Mélanger les responsabilité des étapes

```
@given("qu'un vélo de marque {brand} est en stock")
def step(ctx: Context, brand: str) -> None:
    bike_desc = ctx.docs[brand]
    ctx.current_bike_desc = bike_desc

@when("un utilisateur recherche les vélos de marque {brand}")
def step(ctx: Context, brand: str) -> None:
    assert("current_bike_desc" in ctx)
    ctx.db_client.replace_one(ctx.current_bike_desc, upsert=True)

    ctx.current_bike_doc = ctx.db_client.replace_one(bike_desc, upsert=True)

    url = "/bikes/find-by-brand"
    ctx.resp = ctx.http_client.get(url, params={"brand": brand})
```

👉 Fragile !

- Faire référence aux scénarios dans les implémentations

👉 Fragile !

Conclusion

- Essayez, ça peut réussir !
- ... mais être vigilant
- ... et ne pas perdre de vue les intérêts

Références

- Introducing BDD, Dan North, 2006
- Programmation pilotée par le comportement
- Le Concombre masqué contre le Grand Patatoseur, Nikita Mandryka, Dupuis, 1992



AH MAIS OUI! J'Y SUIS, C'EST ÇA,
C'EST L'HEURE EXQUISE

DE LA SIESTE!!!

HEURE
EXQUISE!

