A LA DÉCOUVERTE DE L'ATDD –
UNE PRATIQUE D'ÉQUIPE POUR LE DÉVELOPPEMENT

Ce qui suit est intégralement extrait de 3 articles du blog Xebia écrits par Grégory Fontaine.
Ces articles sont disponible en ligne aux adresses suivantes s:
https://blog.xebia.fr/2016/12/19/a-la-decouverte-de-latdd-une-pratique-dequipe-pour-le-developpement-13/
http://blog.xebia.fr/2017/01/03/a-la-decouverte-de-latdd-une-pratique-dequipe-pour-le-developpement-23/
https://blog.xebia.fr/2017/01/11/a-la-decouverte-de-latdd-une-pratique-dequipe-pour-le-developpement-33/

Les tutoriels sur l'ATDD ne manquent pas mais ils occultent bien souvent l'aspect collaboratif en ne mettant pas en évidence en quoi l'ATDD modifie le flux de travail et la dynamique de l'équipe, pour se concentrer sur la technique.

Les protagonistes

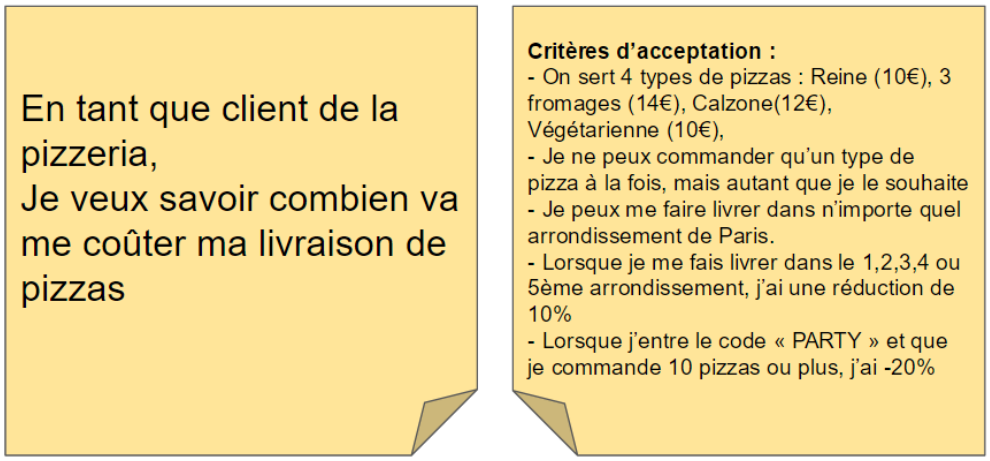
Nous serons accompagnés dans ce tutoriel de 3 protagonistes :

- **Aurore** : **développeur**;
- **Nicolas** : **testeur**, qui font partie de la même équipe Scrum;
- ainsi qu'**Héloïse**, leur **P**roduct **O**wner.

Nicolas est à l'initiative de la démarche d'ATDD dans l'équipe et il s'est donné pour mission de coacher ses collègues dans l'adoption de cette pratique ! Il n'est lui-même pas un expert de la démarche, et il y a fort à parier qu'il tombera dans quelques pièges …

La User Story

L'équipe travaille au développement d'un site web pour une pizzeria qui livre à domicile, dans Paris.
Héloïse soumet à l'équipe la première User Story :



Spécification par l'exemple

Dans une démarche d'ATDD, l'équipe commence par rédiger les tests d'acceptation associés à la User Story. Cela peut être fait lors d'une réunion type « Backlog Refinement Meeting » ou « Backlog Grooming ». Avec toute l'équipe, ou en plus petit comité, en général à 3 : le Product Owner, un testeur et un développeur.
En l'occurrence, Héloïse, Aurore et Nicolas se sont simplement réunis tous les 3, quelques jours avant le prochain Sprint Planning et se sont donnés 15 minutes pour rédiger les cas de tests. La micro-réunion commence :

À LA DÉCOUVERTE DE L'ATDD

« UNE PRATIQUE D'ÉQUIPE POUR LE DÉVELOPPEMENT »

Cas de test	Type de pizza	Quantité	Code promo	Arrondissement	Prix

Cas de test	Type de pizza	Quantité	Code promo	Arrondissement	Prix
1	Reine	1		6ème	10€

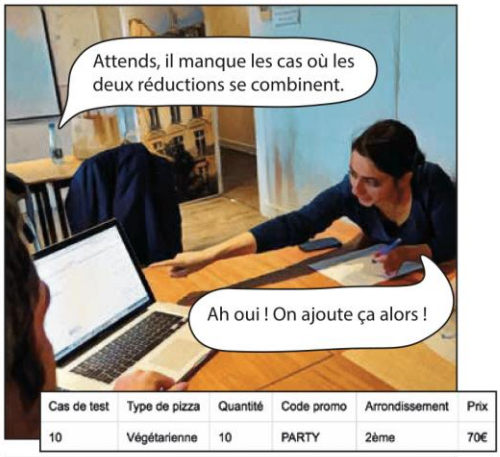
Cas de test	Type de pizza	Quantité	Code promo	Arrondissement	Prix
2	3 fromages	2		13ème	28€

Cas de test	Type de pizza	Quantité	Code promo	Arrondissement	Prix
3	Calzone	2		1er	21.6€

Cas de test	Type de pizza	Quantité	Code promo	Arrondissement	Prix
4	Calzone	2		2ème	21.6€
5	Calzone	2		3ème	21.6€
6	Calzone	2		4ème	21.6€
7	Calzone	2		5ème	21.6€

Cas de test	Type de pizza	Quantité	Code promo	Arrondissement	Prix
8	Végétarienne	10	PARTY	8ème	80€

Cas de test	Type de pizza	Quantité	Code promo	Arrondissement	Prix
9	Végétarienne	9	PARTY	8ème	90€



Cas de test	Type de pizza	Quantité	Code promo	Arrondissement	Prix
10	Végétarienne	10	PARTY	2ème	70€

Voilà ! En 10 minutes chrono, Aurore, Héloïse et Nicolas ont finis de « spécifier par l'exemple » cette User Story. Aurore s'est évité un joli bug sur la méthode de calcul, et Héloïse a retenu que parfois, un exemple valait mieux qu'un long discours. Les 3 collègues mettent donc la User Story de côté pour le moment, et jusqu'au prochain sprint.

La spécification par l'exemple peut aussi avoir lieu plus tard. Dès que la User Story commence par exemple. Plus on fait l'exercice tôt, plus il y a un risque de devoir y revenir, voire de gâchis pur et simple si les priorités changent du tout au tout et que la User Story se retrouve écartée. A l'inverse, en la faisant au dernier moment, vous acceptez de découvrir plus de choses en cours de Sprint, et donc, notamment, d'avoir des estimations un peu moins fiables. A vous de trouver ce qui fonctionne le mieux dans votre équipe !

Dans le prochain article, nous rentrerons dans le vif du sujet et suivrons Nicolas et Aurore, qui ensemble réaliseront leur première User Story en commençant par l'automatisation des tests d'acceptation.

Dans le premier article, nous avons commencé à suivre une petite équipe de développement dans sa découverte de « l'Acceptance Test-Driven Development ». Nous avons quitté Héloïse, Aurore et Nicolas après une première phase de « spécification par l'exemple ». Nous les retrouvons dans le second article de cette série alors qu'ils s'apprêtent à démarrer l'implémentation de leur première User Story !

Automatisation des premiers tests d'acceptation

Le Sprint démarre donc, et la User Story « *En tant que client de la pizzeria, je veux savoir combien va me coûter ma livraison de pizzas* » est en bonne première place dans le Sprint Backlog. Aurore et Nicolas décident de travailler ensemble afin de continuer à expérimenter l'ATDD sur cette User Story.

À LA DÉCOUVERTE DE L'ATDD

« UNE PRATIQUE D'ÉQUIPE POUR LE DÉVELOPPEMENT »

Donc le test donnerait quelque chose comme :

```
Scenario: Commander 1 Reine dans le 6ème arrondissement coûte 10€
  Given I go to "http://..../commande.php"
  And I select "Reine" from "pizza"
  And I fill in "quantity" with "1"
  And I select "6" from "arrondissement"
  When I press "Commander"
  Then I should see "10" in the "#price" element
```

Nicolas lance alors le test et bien sûr, il échoue :

1 test failed

Aurore joue le jeu et implémente donc juste ce qu'il faut pour que le test passe. Dans un modèle « MVC », le code de la Vue pourra ressembler alors à ça :

```
<?php if (isset($this->view['price'])): ?>
    <h1>Prix</h1>
    <span id="price"><?php echo $this->view['price'] ?></span>€
<?php else: ?>
    <form method="POST" action="commande.php">
        Pizza: <select name="pizza">
        <?php foreach($this->view['pizzas'] as $pizza=>$price){ ?>
            <option value="<?php echo $price; ?>"><?php echo $pizza; ?></option>
        <?php } ?>
        </select> x
        <input type="text" name="quantity" size="2"><br />
        Code promo: <input type="text" name="discount"><br />
        Arrondissement pour la livraison: <select name="arrondissement">
        <?php foreach($this->view['arrondissements'] as $arrondissement){ ?>
            <option value="<?php echo $arrondissement; ?>"><?php echo $arrondissement; ?></option>
        <?php } ?>
        </select><br />
        <input type="submit" value="Commander" class="button">
    </form>
<?php endif; ?>
```

Et le code du Modèle pourra lui ressembler à cela :

```php
public function calculateCommande($pizza, $quantity, $discount, $arrondissement)
{
    return 10;
}
```

Je pense qu'il existe de nombreuses façons de collaborer autour de l'ATDD et d'en tirer un bénéfice. Ici, le testeur et le développeur restent ensemble, autour d'une même machine. Mais d'autres organisations sont possibles. Nicolas pourrait être à son ordinateur, Aurore au sien, et tous deux pourraient committer leur code (de test pour l'un, de production pour l'autre) dans une même branche (et pourquoi pas le trunk/master). Une autre organisation possible bien sûr, serait que Nicolas laisse Aurore implémenter les tests automatisés toute seul !

De même, dans cet exemple, Nicolas et Aurore vont implémenter les tests un à un, en s'interdisant d'écrire un test supplémentaire tant que les précédents ne passent pas tous avec succès. Mais, tout en gardant à l'esprit de les faire passer au vert les uns après les autres, on pourrait imaginer les écrire tous à l'avance.


```gherkin
Scenario Outline: Commander des pizzas
  Given I go to "http://..../commande.php"
  And I select "<pizza>" from "pizza"
  And I fill in "quantity" with "<quantity>"
  And I select "<arrondissement>" from "arrondissement"
  When I press "Commander"
  Then I should see "<price>" in the "#price" element
Examples:
| pizza       | quantity | arrondissement | price |
| Reine       | 1        | 6              | 10    |
| 3 fromages  | 2        | 13             | 28    |
```

Ils lancent les tests et obtiennent :

1 test passed
1 test failed

Aurore fait le nécessaire pour faire passer ce deuxième test au vert. Pas grand-chose en l'occurrence. Juste une multiplication :

```
public function calculateCommande($pizza, $quantity, $discount, $arrondissement)
{
    return $pizza * $quantite;
}
```

Puis ils relancent les tests : 2 tests passed

Et ainsi de suite, jusqu'à ce que Nicolas et Aurore aient implémenté les 10 cas de tests, et le code permettant de les faire passer avec succès. Le code de test final ressemble alors à ceci :

```
Scenario Outline: Commander des pizzas
  Given I go to "http://..../commande.php"
  When I select "<pizza>" from "pizza"
  And I fill in "quantity" with "<quantity>"
  And I fill in "discount" with "<discount>"
  And I select "<arrondissement>" from "arrondissement"
  And I press "Commander"
  Then I should see "<price>" in the "#price" element
  Examples:
    | pizza       | quantity | discount | arrondissement | price |
    | Reine       | 1        |          | 6              | 10    |
    | 3 fromages  | 2        |          | 13             | 20    |
    | Calzone     | 2        |          | 1              | 21.6  |
    | Calzone     | 2        |          | 2              | 21.6  |
    | Calzone     | 2        |          | 3              | 21.6  |
    | Calzone     | 2        |          | 4              | 21.6  |
    | Calzone     | 2        |          | 5              | 21.6  |
    | Végétarienne| 10       | PARTY    | 8              | 80    |
    | Végétarienne| 9        | PARTY    | 8              | 90    |
    | Végétarienne| 10       | PARTY    | 2              | 72    |
```

Et la méthode calculateCommande du modèle pourra ressembler à cela :

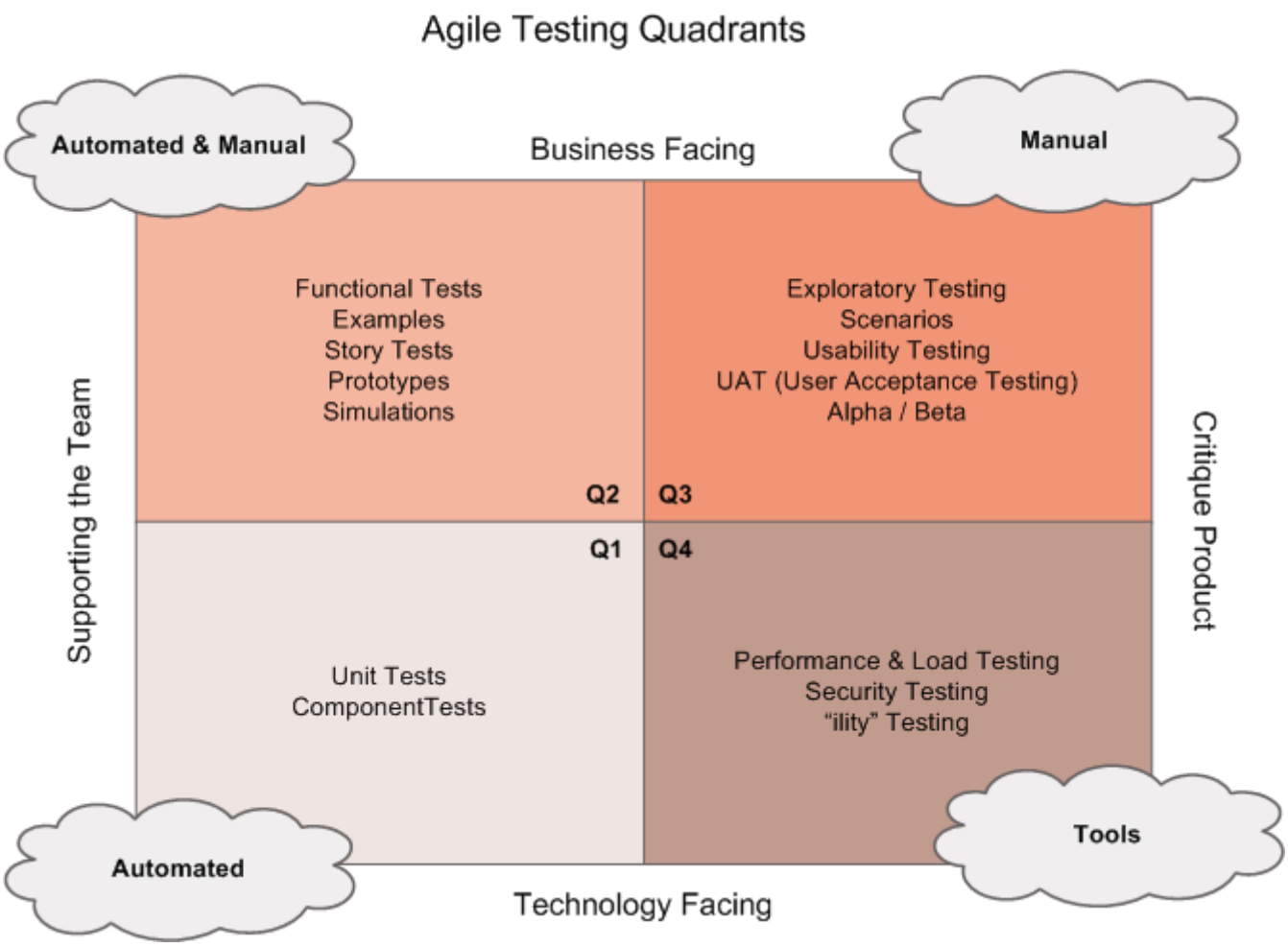
```
public function calculateCommande($pizza, $quantity, $discount, $arrondissement)
{
    $price = $pizza * $quantity;

    $closestArrondissements = array(1,2,3,4,5);
    if (in_array($arrondissement,$closestArrondissements))
        $price = $price * 0.9;

    if ($discount == "PARTY" && $quantity >= 10)
        $price = $price * 0.8;

    return $price;
}
```

Et oui, l'ATDD n'a pas la prétention de remplacer tous vos tests par des tests automatisés. Si l'on reprend les célèbres quadrants, on remarque que les tests que l'on a écrits ne concernent que la partie supérieure gauche !



En revanche, puisque l'ATDD cherche à rendre utilisable une partie au moins de la fonctionnalité beaucoup plus tôt qu'avec une approche plus traditionnelle du développement, elle permet de commencer à tester d'autres aspects bien plus tôt aussi.

C'est tout pour ce deuxième article de la série. Mais ne vous arrêtez surtout pas là ! Car, certains d'entre vous l'ont peut-être remarqué, Nicolas et Aurore ont commis de sérieuses erreurs dans leur première expérience avec l'ATDD. Dans le prochain article, nous verrons ce qui cloche exactement dans ce qu'ils ont fait, quelles conséquences cela a sur l'équipe à long terme, et apporterons des solutions.

A très vite !

Nous retrouvons Aurore, Nicolas, Héloïse, et Emma, la petite nouvelle, pour le troisième et dernier article de la série sur l'Acceptance Test-Driven Development. Nous y verrons les problèmes que peuvent poser l'ATDD sur la durée et réparerons les quelques erreurs de jeunesse commises par Aurore et Nicolas dans l'article précédent. Bonne lecture !

3 MOIS PLUS TARD ...

3 mois plus tard, le produit a évolué. De nouvelles fonctionnalités sont venues se rajouter par-dessus la première. L'équipe d'Aurore et de Nicolas est désormais accro à l'ATDD. Ils ont développés 200 tests d'acceptation, inspirés des premiers, et qui tournent à chaque commit. Nicolas est un testeur comblé, il y a moins de bugs dans le produit final, et le temps de cycle a été réduit de 30%.

Héloïse arrive alors au Grooming avec une idée "révolutionnaire".

Quelques jours plus tard, le Sprint avec cette user Story commence donc et Aurore se jette dessus, comme à son habitude.


```
And I select "<pizza>" from "pizza"
And I fill in "quantity" with "<quantity>"
And I fill in "discount" with "<discount>"
And I select "<arrondissement>" from "arrondissement"
When I press "Commander"
```

par une unique instruction :

```
When I order "<quantity>" x "<pizza>" for the "<arrondissement>"th arrondissement with discount code "<discount>"
```

Aurore implémente elle le code qui va traduire cette nouvelle instruction Gherkin, en une série d'instructions pour le navigateur. Avec Behat, cela consisterait à ajouter ceci au « contexte » :

```
/**
 * Orders specified quantity of specified pizza for the specified arrondissement with the specified discount code
 * Example: Given I order "1" x "Reine" for the "6"th arrondissement with discount code ""
 * Example: When I order "15" x "3 fromages" for the "1"th arrondissement with discount code "PARTY"
 *
 * @When /^(?:I )order "(?P<quantity>(?:[^"]|\\")*)" x "(?P<pizza>(?:[^"]|\\")*)" for the "(?P<arrondissement>(?:[^"]|\\")*)"th arrondissement with discount code "(?P<discount>(?:[^"]|\\")*)"$/
 */
public function orderPizzas($quantity, $pizza, $arrondissement, $discount)
{
    $this->selectOption("pizza", $pizza);
    $this->fillField("quantity", $quantity);
    $this->fillField("discount", $discount);
    $this->selectOption("arrondissement", $arrondissement);
    $this->pressButton("Commander");
}
```

6 MOIS PLUS TARD ...

Encore 3 mois ont passés. Le produit a encore évolué et s'est étoffé. L'équipe a désormais 500 tests d'acceptation et Nicolas est aux anges... Mais certains développeurs un peu moins.

L'ensemble des tests automatisés (unitaires, d'acceptation, etc.) prends désormais 1 heure. Sur ces 60 minutes, 50 sont passées à exécuter les tests d'acceptation écrit avec Behat. La rétrospective s'annonce animée, notamment vis-à-vis d'Emma, qui commence à ouvertement rendre Behat, et l'ATDD par la même occasion, responsable de la baisse de vélocité du dernier sprint.

La rétrospective se termine, et l'équipe s'engage sur ces actions : séparer les tests en deux, et mettre de côté certains tests d'acceptation.

L'optimisation des tests automatisés et de l'intégration continue est un puit sans fond dans lequel les équipes qui pratiquent l'amélioration continue peuvent puiser régulièrement. Cela donne souvent d'excellentes actions.

En passant en revue les tests d'acceptation, Nicolas s'arrête sur les 10 premiers tests écrit 6 mois auparavant.

Il remarque 5 tests qui ne diffèrent que par l'arrondissement sélectionné. Il se souvient vaguement d'une discussion avec Héloïse et Aurore autour de la pertinence d'ajouter ces tests.

Mais aujourd'hui, il est clair qu'ils n'apportent pas grand-chose car il existe de nombreux autres tests, parmi les 500, qui permettent de s'assurer que la réduction s'applique bien au 1er, 2ème, 3ème, 4ème et 5ème arrondissement. Nicolas en garde donc un, et efface les autres.

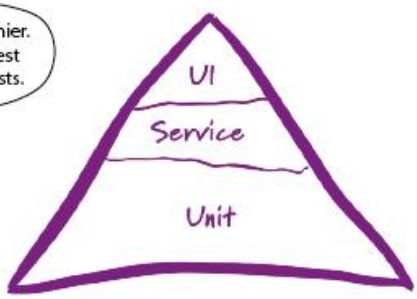
Tous les tests que vous écrivez au moment de la spécification par l'exemple n'ont pas vocation à être automatisés ! De même : tous les tests que vous automatisez pendant le développement d'une User Story car ils vous guident dans l'implémentation de la fonctionnalité n'ont pas vocation à être ensuite intégrés à vie dans une suite de tests de non-régression.

Il applique le même raisonnement sur les 500 tests et parvient à réduire leur nombre à 350 ! Emma retrouve le sourire, et tout rendre dans l'ordre. L'équipe conclut aussi qu'il faudra désormais être un peu plus regardant sur les tests d'acceptation qu'ils automatisent.

9 MOIS PLUS TARD ...

L'équipe a continué à pratiquer l'ATDD et a pris soin de ne pas ajouter trop de tests. Mais ils se sont inexorablement rapprochés des 500 tests, et 3 mois plus tard, les voilà de retour au même point ! Cette fois-ci, à la rétrospective, Emma veut la peau de Behat et de l'ATDD.

La rétrospective se termine et l'équipe s'engage à réécrire certains tests d'acceptation afin qu'ils ne passent pas par l'interface utilisateur. Nicolas et Aurore se retrouvent le lendemain pour s'attaquer au problème.



Pyramide des tests

Ce qu'on a nous dans l'équipe, ce n'est pas une pyramide, c'est un sablier ! Il faut qu'on redescende certains de nos tests écrits avec Behat d'un niveau.

Il est tout à fait possible d'écrire un test au niveau du service, d'une API, à l'aide d'un framework dit de "test unitaire". C'est bien souvent la solution la plus simple techniquement pour le développeur. Mais attention, cela peut compromettre sérieusement la capacité à utiliser les tests comme support de discussion avec le testeur et le Product Owner.

Imaginons ici que le code a évolué et que la commande de pizza soit désormais possible aussi via une API par exemple. C'est cette API qui sera sollicitée par les tests. Aurore reprends donc les appels à l'interface utilisateur (remplir un champs, choisir une valeur dans une liste déroulante, cliquer sur un bouton, etc.) qui, rappelez-vous, avaient été regroupés dans le "contexte" de Behat, et les remplace par un appel à l'API. Au final, cela représente peu de changement de code et dans la journée, Aurore a terminé !


OK pour une douzaine de test. Je m'occupe de sélectionner les scénarios les plus représentatifs et les plus critiques et de les écrire comme avant.


À l'avenir, ce qu'on fera à chaque fois qu'on fera de l'ATDD sur une Story, c'est de réfléchir à la meilleure manière de tester, en gardant à l'esprit d'être de plus en plus regardant avec ce qu'on automatise à mesure que l'on monte dans la pyramide.


Aurore, laisses moi te dire que tu déchires !
Merci, toi aussi !

FIN

Conclusion

A travers cette histoire, j'espère avoir illustré les changements que l'ATDD engendre dans le travail de l'équipe. Ce n'est pas qu'une simple « technique » que l'on applique. C'est une approche complexe, qui nécessite de se poser les bonnes questions, et de se les poser tous ensemble. N'espérez cependant pas avoir tout bon du premier coup. Comme l'ont fait Héloïse, Aurore, Emma et Nicolas, le mieux reste de démarrer, d'observer les résultats, et d'adapter la pratique à son contexte !

L'auteur de ces articles, publiés sur le blog de Xebia, est : Gregory Fontaine