acpqualife
HPS Group

Gratter DéDé

Le xDD démystifié :

Tout ce que vous avez voulu savoir sur DéDé sans jamais oser le demander

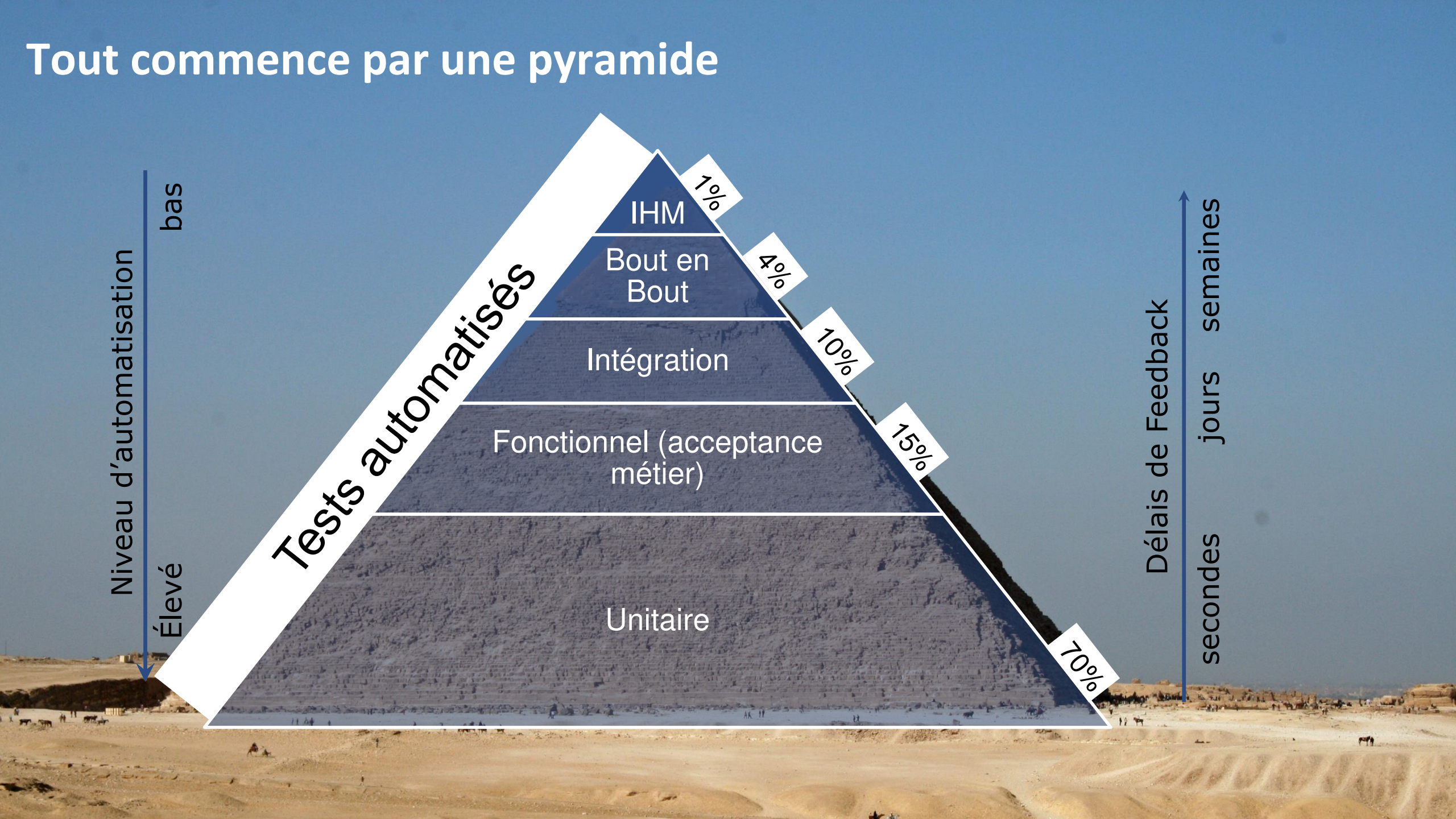
Cyril TARDIEU
cytardieu cyril-tardieu c.tardieu@acpqualife.com

Laurent BOUHIER
laurent_bouhier laurent-bouhier l.bouhier@acpqualife.com

DéDé expliqué :

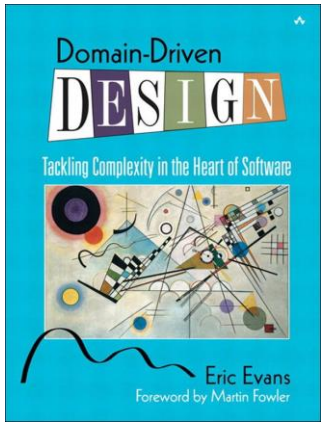
DéDé ?

Tout commence par une pyramide



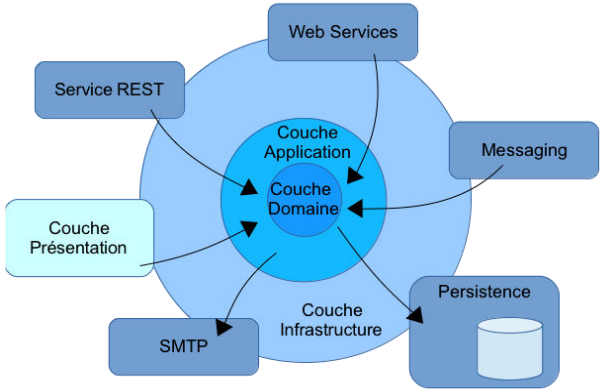
DéDé ?

DDD: Domain Driven Design



« Conception pilotée par le domaine »

- La connaissance du *domaine* métier (modèle) dans un contexte limité
- *Langage omniprésent (« ubiquitous language »)*
- La collaboration entre les développeurs et les experts du *domaine*



Architecture en couches : Préserver la couche domaine (« bounded context »)

Intérêts

- Communication avec les experts métier
- Modèle modulaire et maintenable
- Amélioration de la **testabilité** et de la généricité des objets du *domaine* métier.

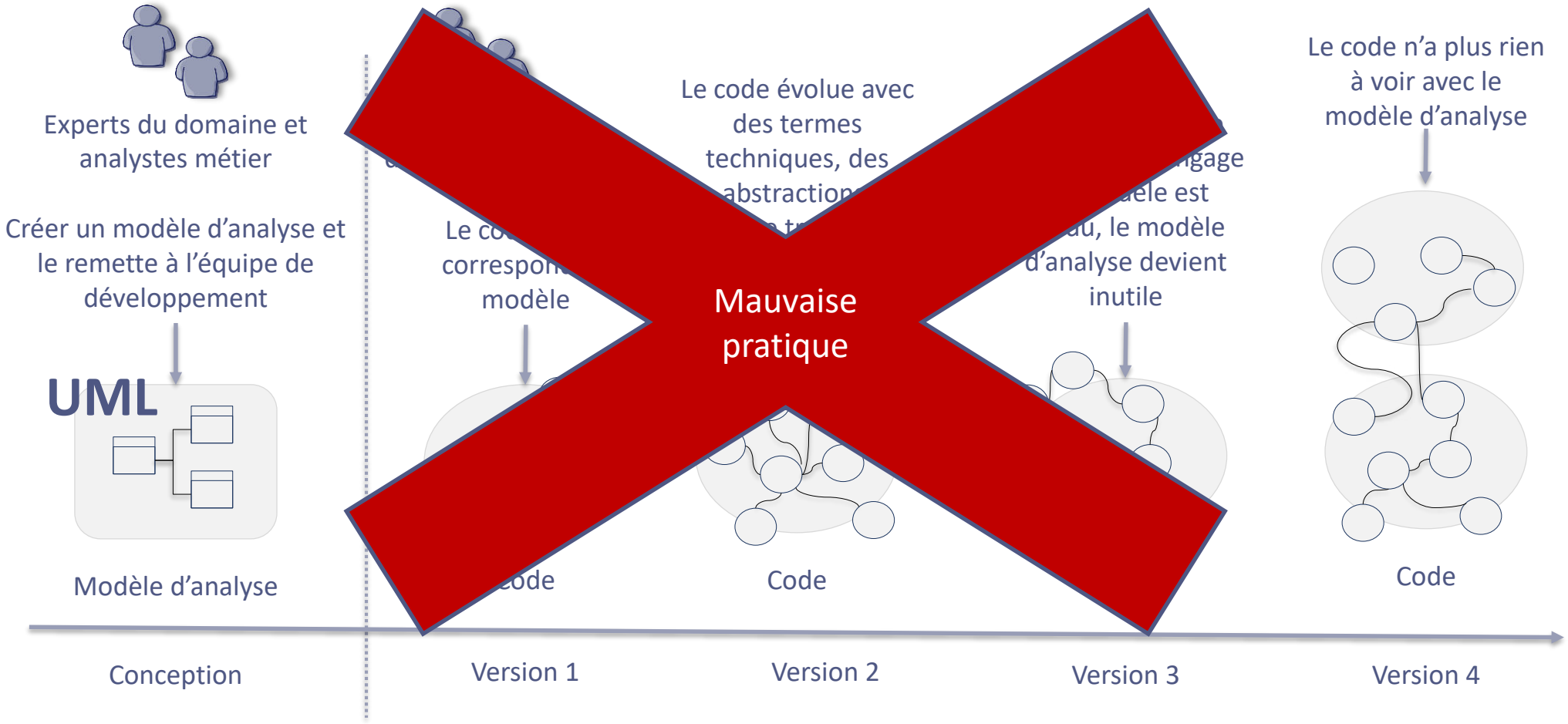
Points de surveillance

- Difficile de convaincre les acteurs métier
- Apprentissage du langage omniprésent et du domaine
- Représenter le « domain model » et garder le couplage avec le code (« software craftsmanship »)

DDD pour éviter une conception en amont

acpqualife HPS Group

Conception « classique » en amont



DDD : Agile et Itératif

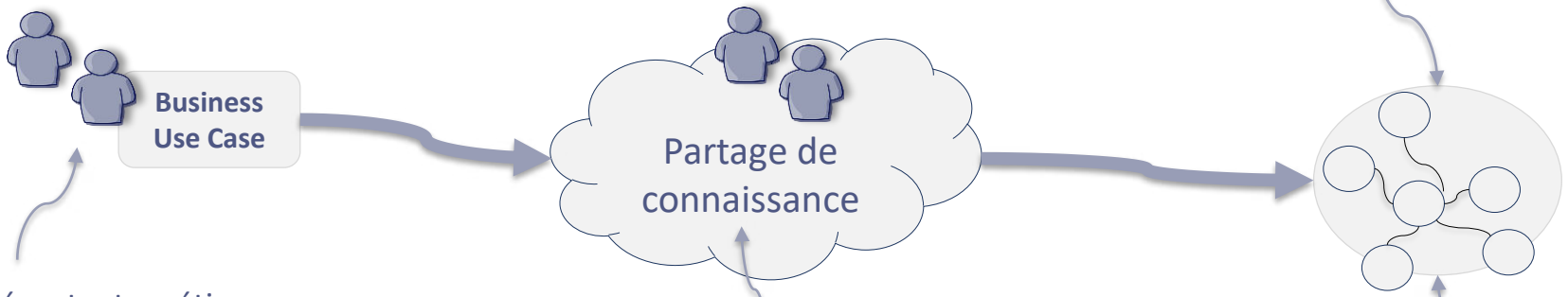
acpqualife HPS Group

Le processus DDD

Représentants métier
&
Equipe de développement

Experts du Domaine
&
Equipe de développement

Simplification du modèle quand on comprend mieux la problématique du domaine



Les représentants métier communiquent les objectifs métier et les entrées et sorties du système. L'équipe capture les « business use cases »

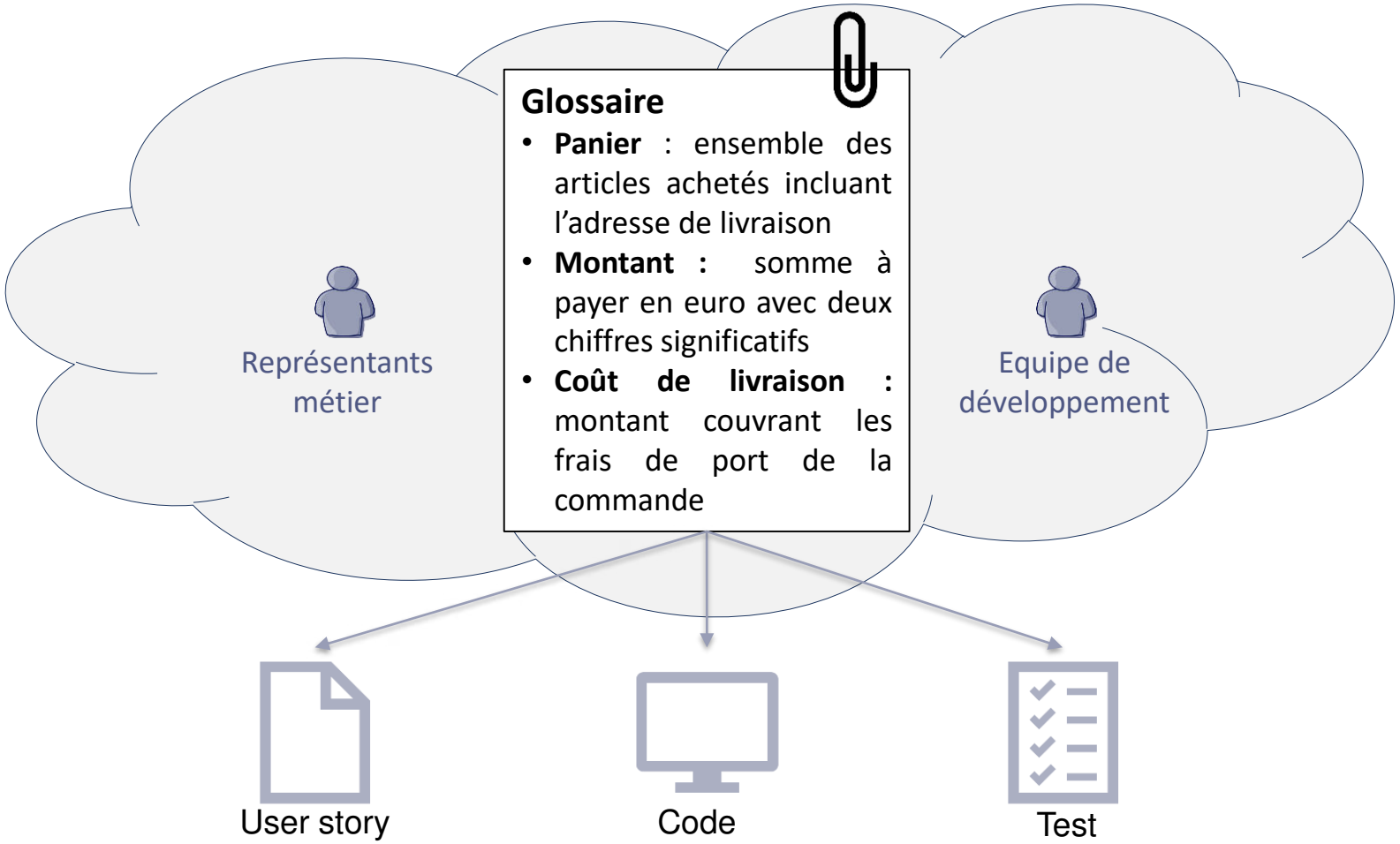
L'équipe de développement et les experts du Domaine collaborent pour produire un modèle qui satisfait les besoins du « business use case »

Un modèle utile et qui satisfait le cas d'utilisation

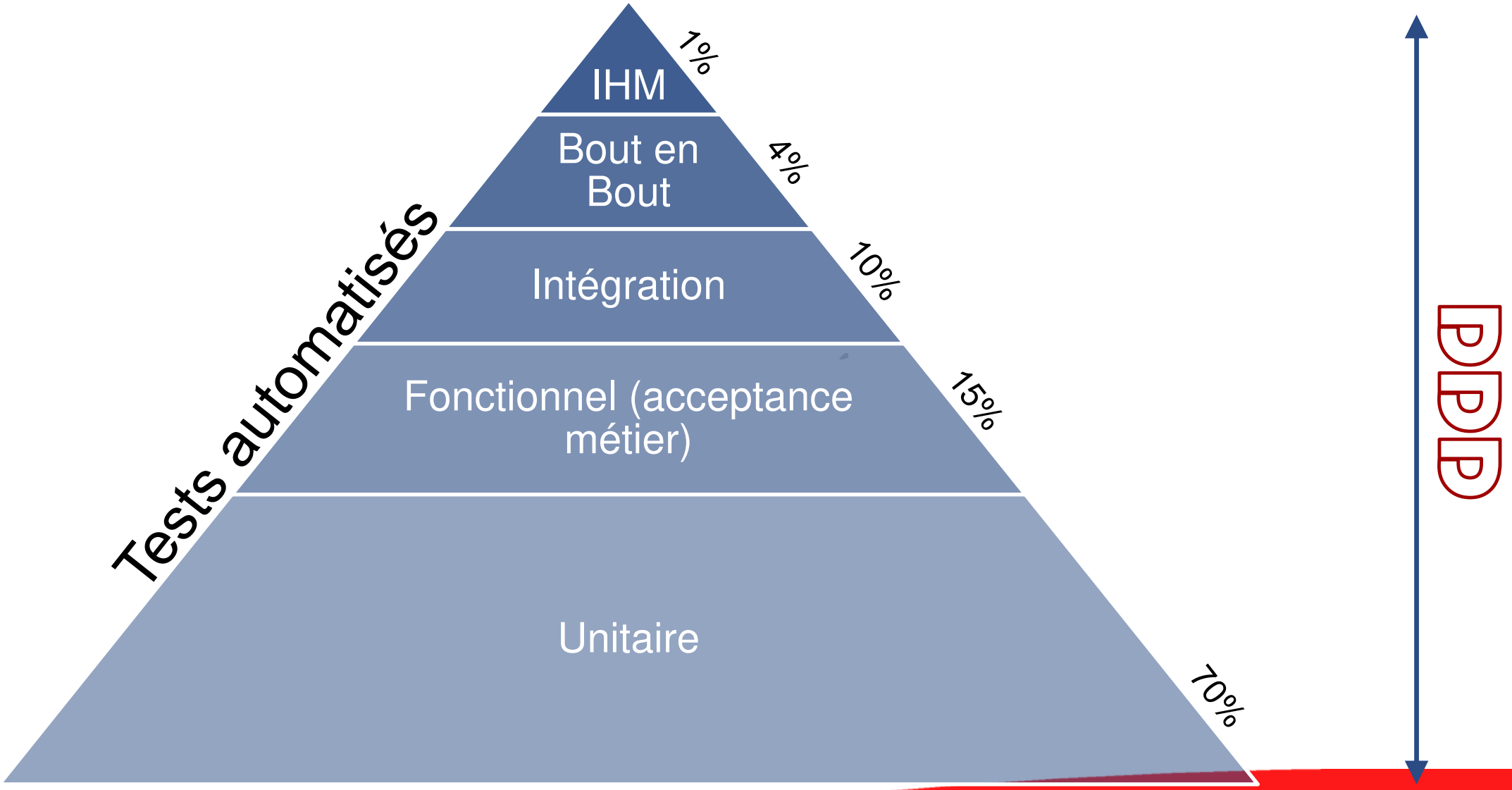
Une équipe, un langage, un modèle

Acpqualife - Reproduction interdite - 2017

DDD : Le langage omniprésent



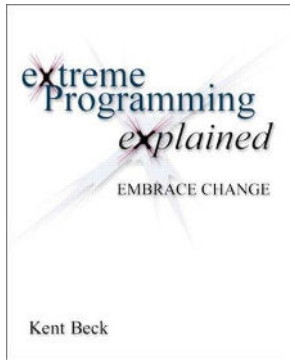
Le DDD sur la pyramide



DéDé ?

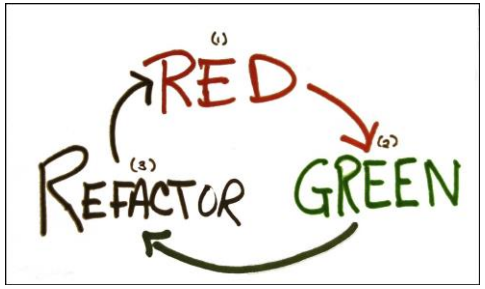
TDD: Test Driven Development

acpqualife
HPS Group



« Développement piloté par les Tests »

- Origine : extreme programming (XP) 1999
- Ecrire le test avant le code
- Concept de tester tôt (test first)
- Processus de développement à part entière – pilote la conception !



TDD
=
Tests unitaires
=
Code

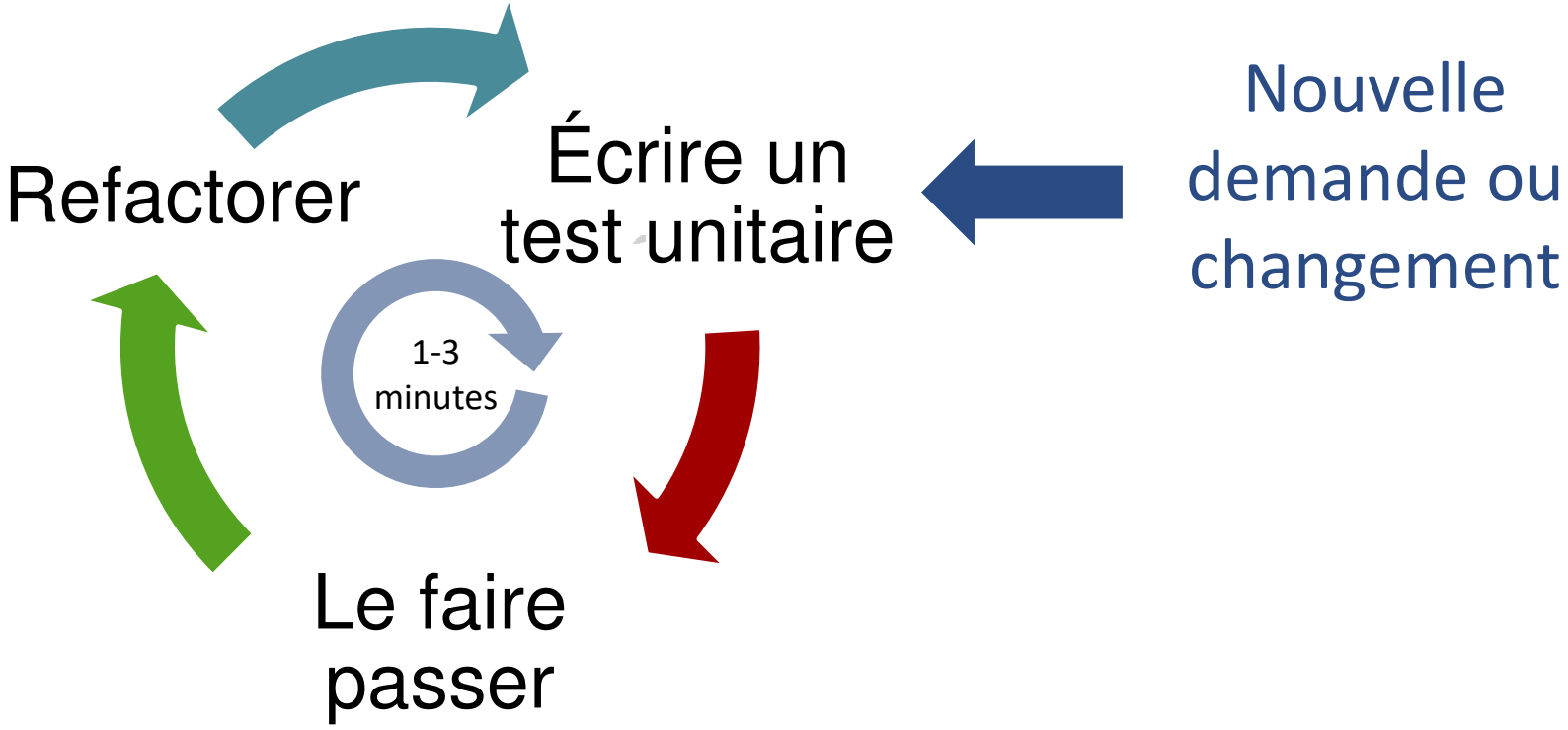
Intérêts

- Assure une couverture technique
- Améliore la testabilité du code (il est conçu pour être testable)
- Maintenabilité (refactoring) accrue
- Augmente la confiance dans le code

Points de surveillance

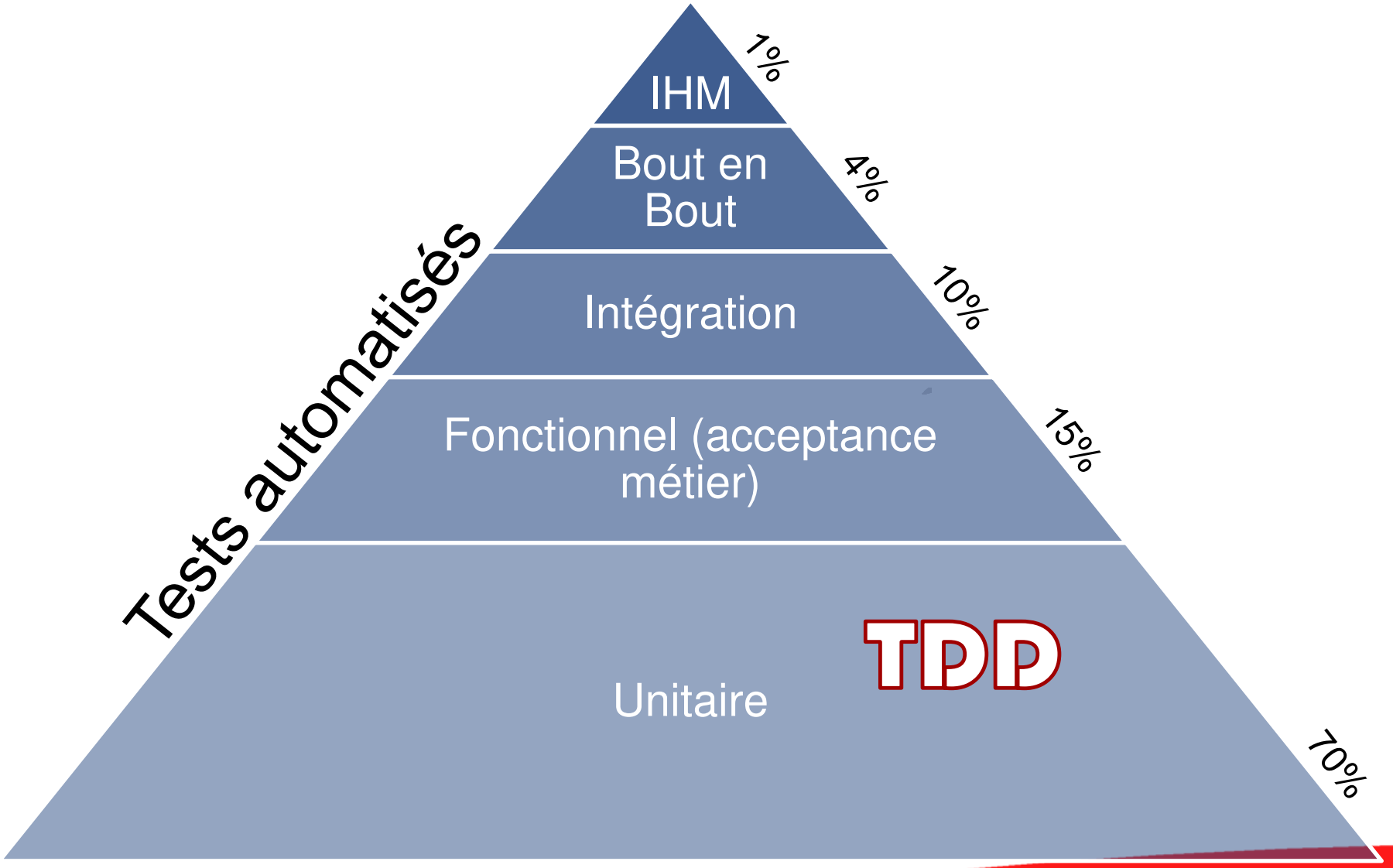
- Avoir des développeurs formés
- KISS (rester simple !)
- Ne pas se fier uniquement aux tests unitaires (pyramide !)
- Maintenir les tests (intégration continue)

TDD: Test Driven Development



Acpqualife - Reproduction interdite - 2017

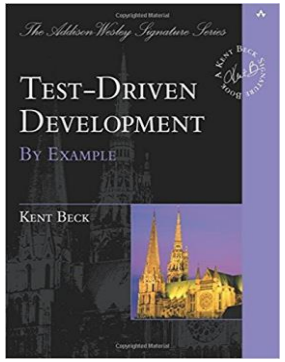
Le TDD sur la pyramide



DéDé ?

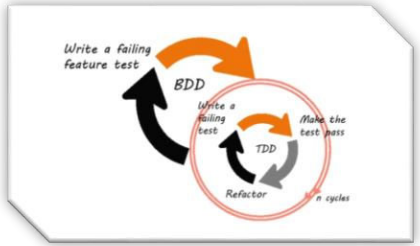
ATDD: Acceptance Test Driven Development
BDD: Behavior Driven Development

acpqualife
HPS Group



« Développement piloté par les Tests d'Acceptation»

- Origine : "Test Driven Development: By Example" 2003
- Concept de tester tôt (test first)
- Acceptance = critère d'acceptance de la user story !



- Double boucle TDD/ATDD
- Ecrire les cas de tests de story avant le développement
- Outillage : cucumber (BDD), fitness…
- Bien adapté avec du DDD (ubiquitous language)

Intérêts

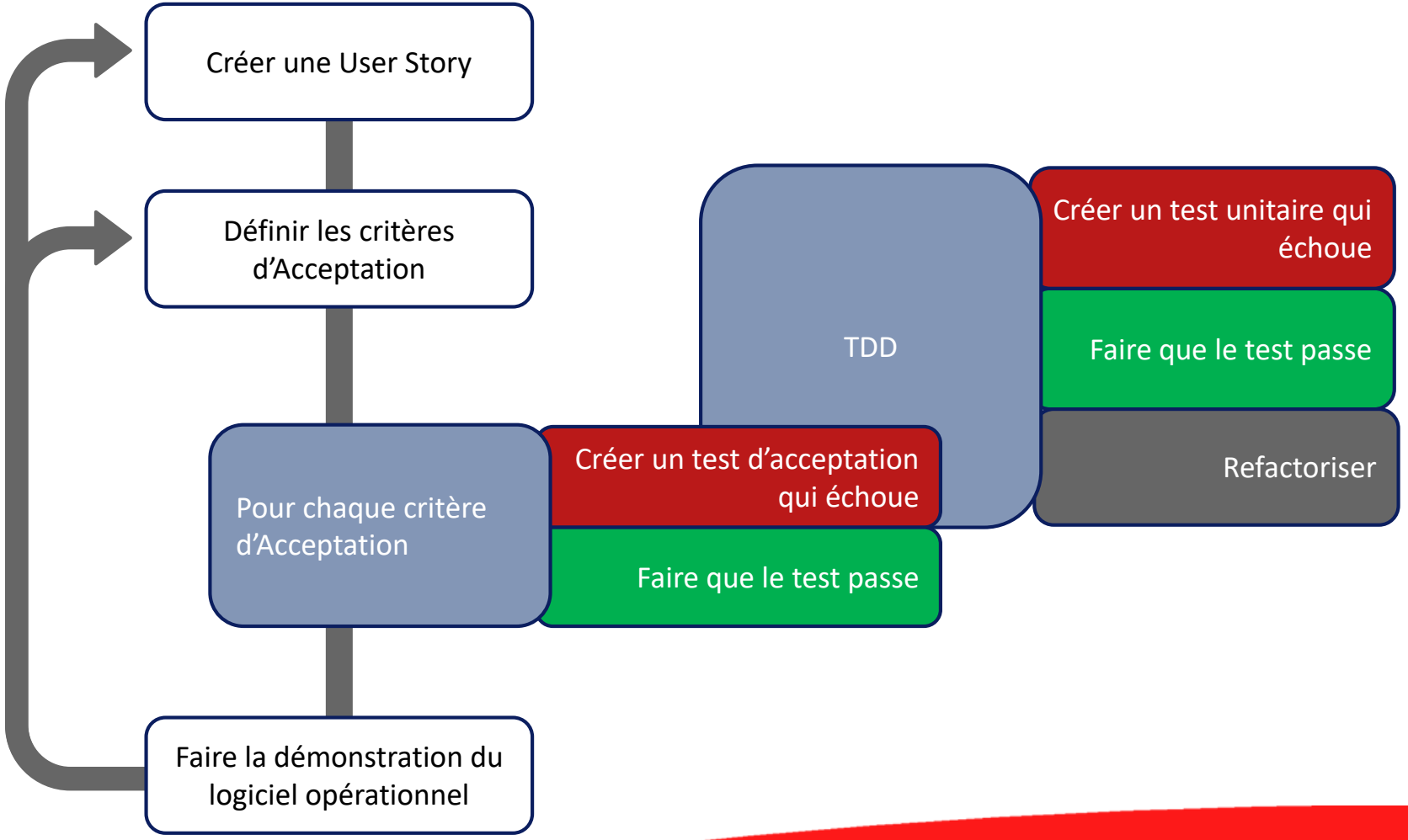
- Tests automatisés par conception (régression)
- Comme pour le TDD : améliore la testabilité et la maintenabilité
- Meilleure communication et vrais critères d'acceptation

Points de surveillance

- Doit vérifier le processus métier (pas que l'IHM…)
- Attention à la mauvaise utilisation des outils (ils doivent faciliter le processus pas le piloter !)
- Maintenir les tests (intégration continue)

Acpqualife - Reproduction interdite - 2017

Double boucle ATDD/TDD



BDD = Gherkins – une façon de faire de l'ATDD

acpqualife HPS Group

1. Write story

Plain text

Scenario: A trader is alerted of status

Given a stock and a threshold of 15.0
When stock is traded at 5.0
Then the alert status should be OFF
When stock is traded at 16.0
Then the alert status should be ON

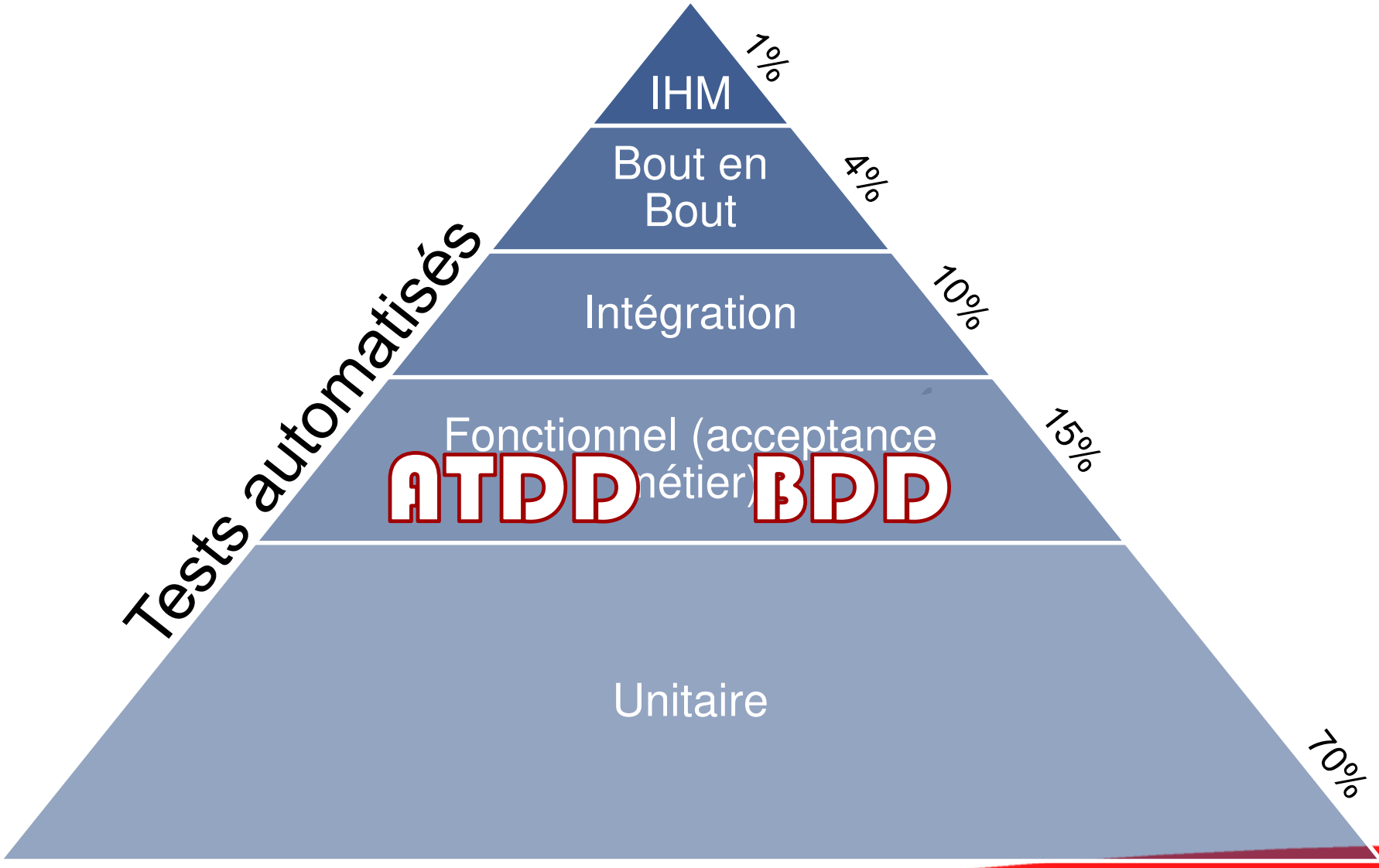
2. Map steps to Java

POJO

```
public class TraderSteps {
    private TradingService service;   // Injected
    private Stock stock; // Created

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

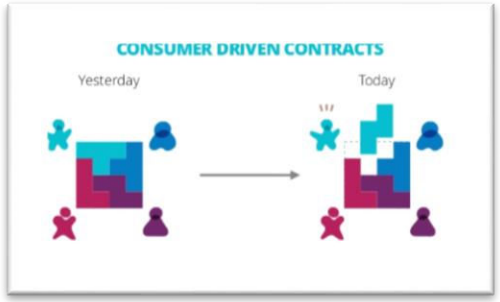
Le ATDD et BDD sur la pyramide



DéDé ?

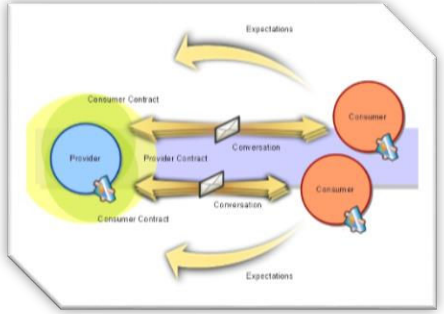
CDD: Contract Driven Development
CDC: Consumer Driven Contract

acpqualife
HPS Group



« Contrat piloté par les consommateurs»

- Le contrat d'interface est fourni par chaque client
- Principalement pour les technologies micro-services (REST)



Un contrat est un accord entre un consommateur et un fournisseur qui décrit les interactions qui peuvent avoir lieu entre eux

Intérêts

- Maintenabilité des interfaces accrue (Postel Law)
- Simplification des tests (bouchon et pilote)
- Maîtriser ses consommateurs
- Services pilotés par la valeur métier

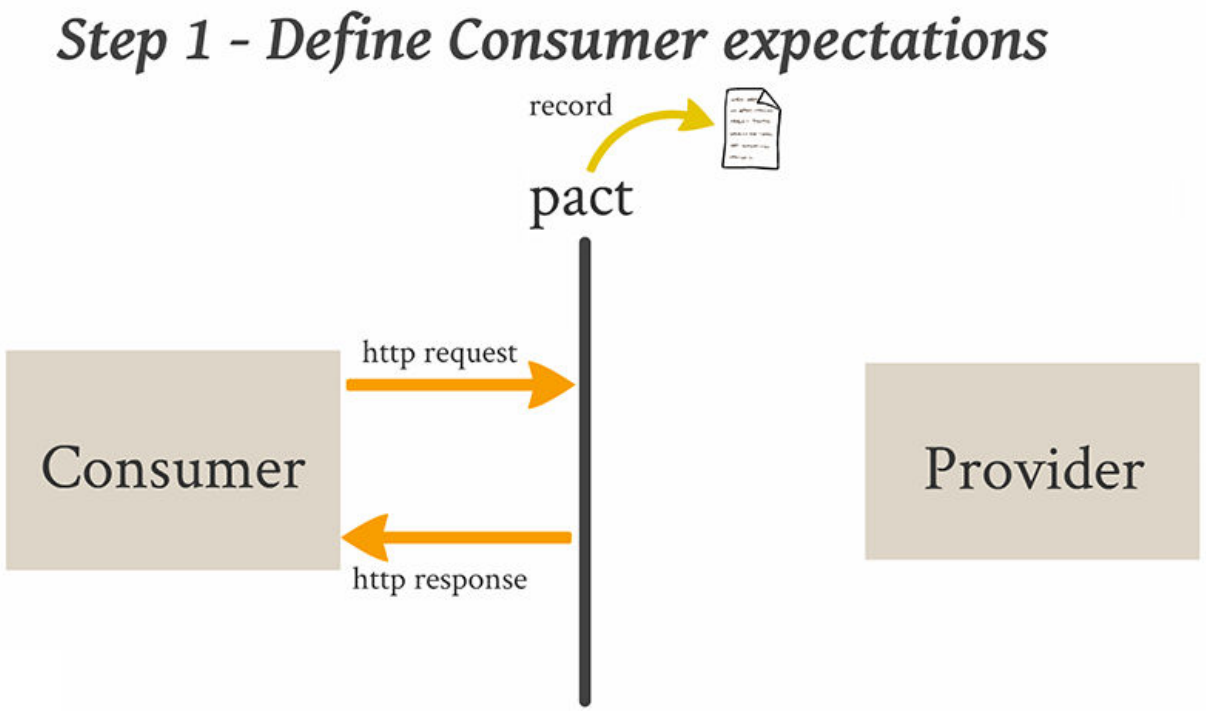
Points de surveillance

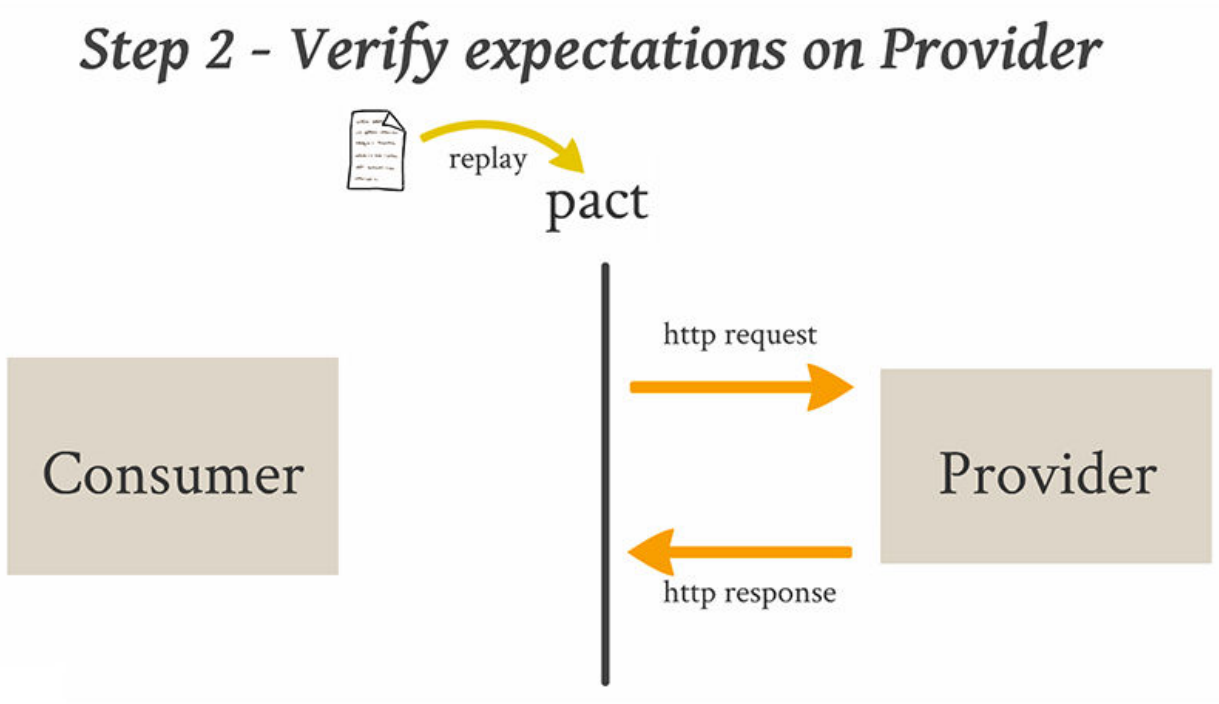
- Attention à l'intégrité des fournisseurs (les contrats des consommateurs doivent être raisonnables)
- Doit être une règle d'entreprise (éviter les appels « pirates » aux services)
- Ne teste que la syntaxe des contrats

Acpqualife - Reproduction interdite - 2017

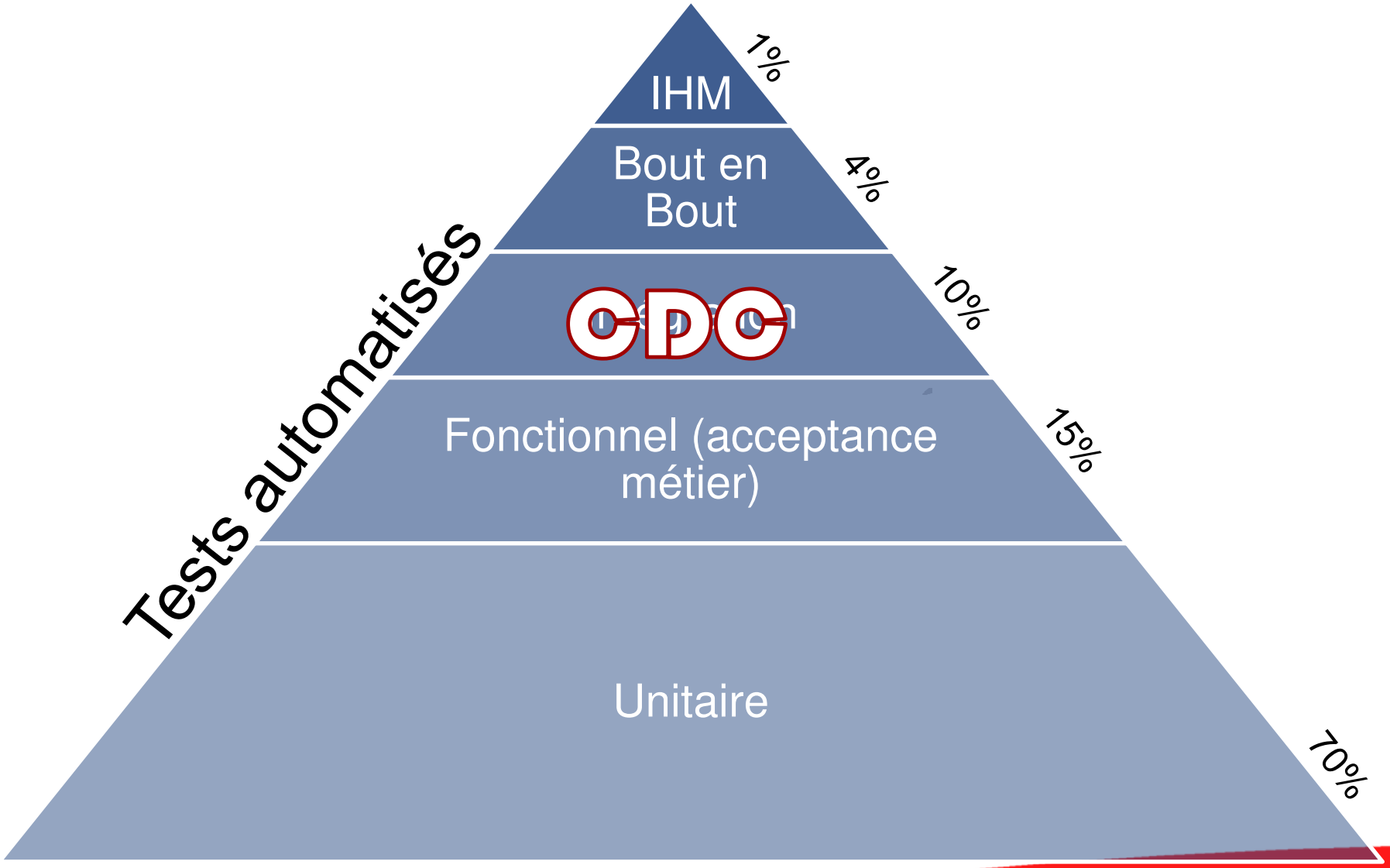
Pact: un outil de vérification de vos contrats !

acpqualife
HPS Group





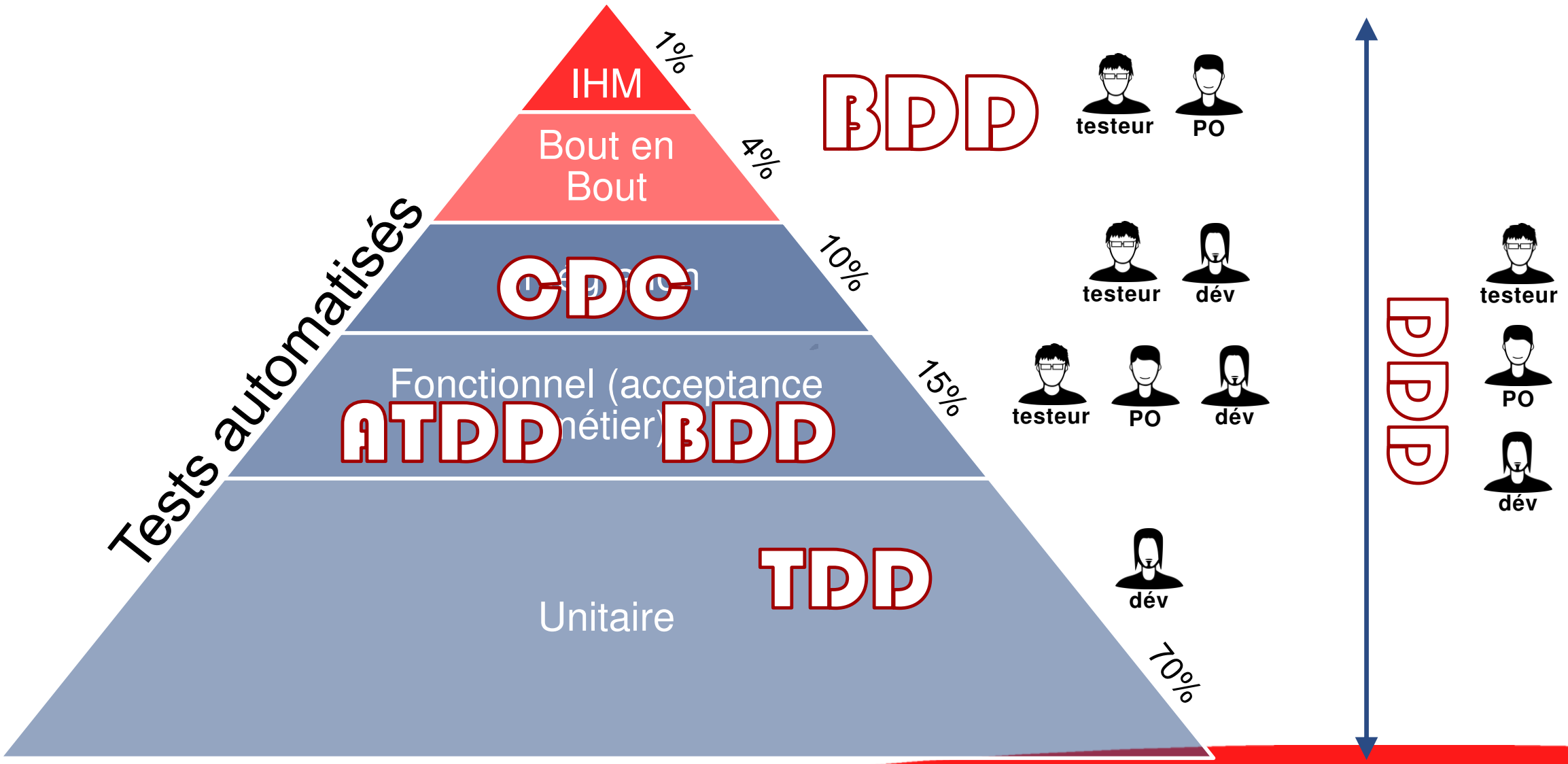
CDC sur la pyramide



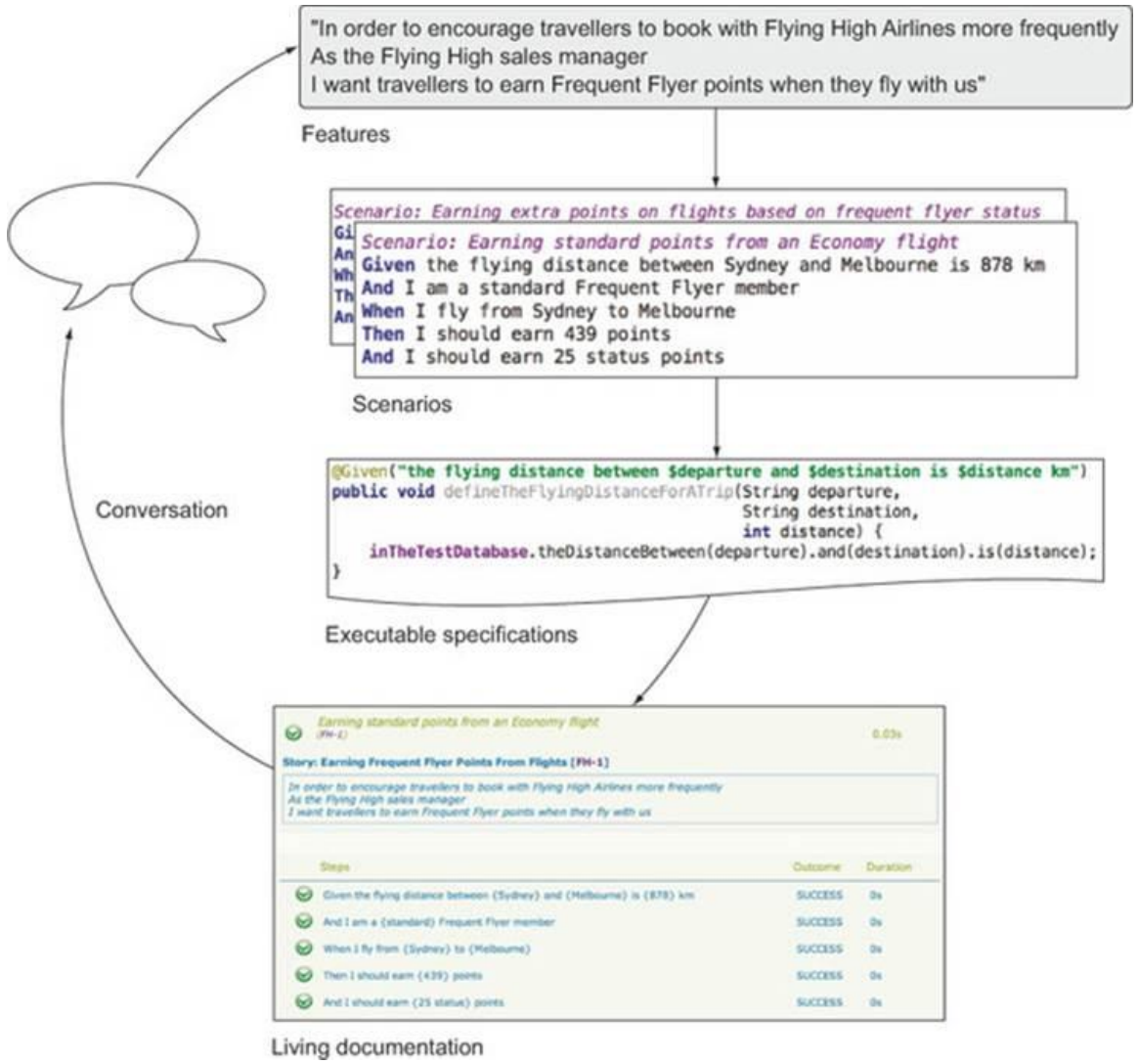
Et ?

Acpqualife - Reproduction interdite - 2017

Une pyramide ?



BDD = « Living documentation »



Peut être automatisé à plusieurs niveaux :

1. « code » : vérification de la règle métier
2. API : vérification du flot
3. IHM

➔ Mais en respectant la pyramide

Example pratique

Fonctionnalité:
Calcul du prix total d'un panier en appliquant les règles :

Fonctionnalité:
Calcul du prix total d'un panier en appliquant les règles :
* TVA 20%
* Coût de livraison 3€ jusqu'à 20€, 2€ sinon

Plan de Scénario: Coût total d'un panier
Étant donné un produit à <prix>€
Quand je l'ajoute au panier
Alors le coût total du panier doit être de <total>€
Exemples:

prix	total
15.00	21.00
25.00	32.00
20.00	26.00

```
public class CostStepdefs implements En {
  public CostStepdefs(SharedDriver webDriver) {
     Given("^un produit à {bigdecimal}€$", (BigDecimal prix) ->
          DBHelper.ajouterProduitDB("Dummy", prix));
     When("^je l'ajoute au panier$", ()-> {
        // cherche le produit Dummy précédemment créé
        webDriver.findElement(By.id("search")).sendKeys("Dummy");
        // ajoute le résultat au panier
        webDriver.findElement(By.id("find")).click();
     });
     Then("^le coût du panier doit être de {string}€$",
     ( String total) ->
     assertEquals(total,
     webDriver.findElement(By.id("total")).getAttribute("value")));
  }
}
```

Règles d’or du haut de la pyramide

AUSSI PEU DE TESTS DE BOUT EN BOUT QUE POSSIBLE : SCENARIOS CRITIQUES

AUTOMATISER EN FAISANT ABSTRACTION DE LA COUCHE DE PRESENTATION

Serenity BDD

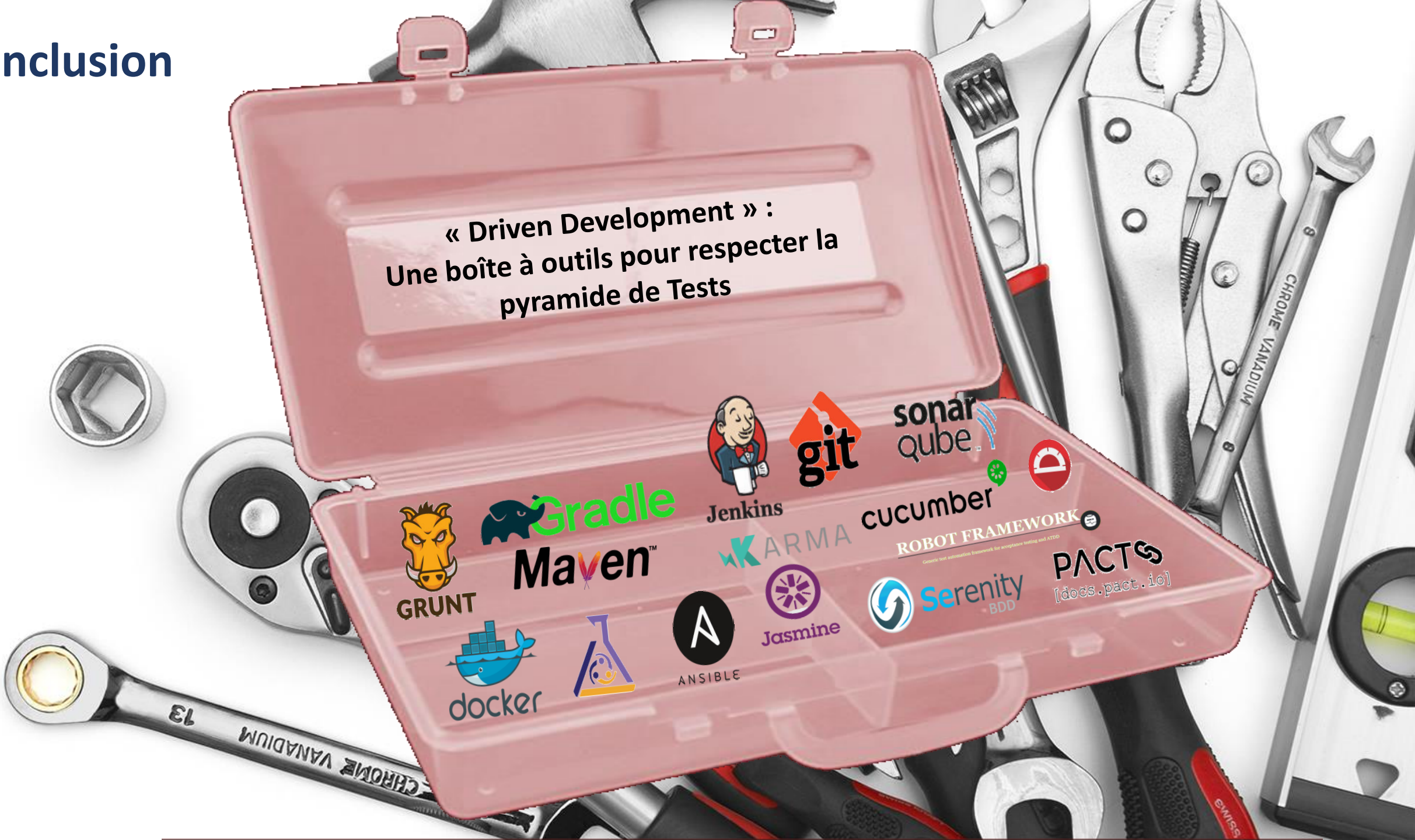
ANSIBLE

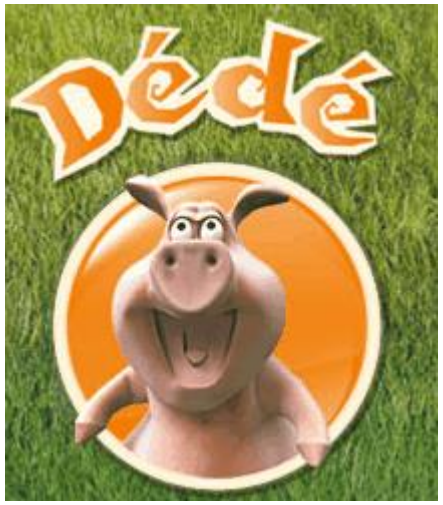
docker

UTILISER DES ENVIRONNEMENTS ET DES DONNÉES DÉDIÉS

COMPLÉTER PAR DU VRAI TEST EXPLORATOIRE

Conclusion


« Driven Development » :
Une boîte à outils pour respecter la pyramide de Tests
GRUNT
Gradle
Maven
Jenkins
git
sonar qube
KARMA
cucumber
ROBOT FRAMEWORK
PACTS
[docs.pact.io]
Serenity BDD
Jasmine
ANSIBLE
docker
CHROME VANADIUM
13

acpqualife
HPS Group

Merci