
ippon
POSITIVE
TECHNOLOGY
Sunny
TECH
Property-based Testing
Ou l'insuffisance des tests unitaires

$ whoami

 # 7 ans d'XP Java back, retail

 # 2 ans chez Ippon Technologies (Lille) Decathlon, PayFit

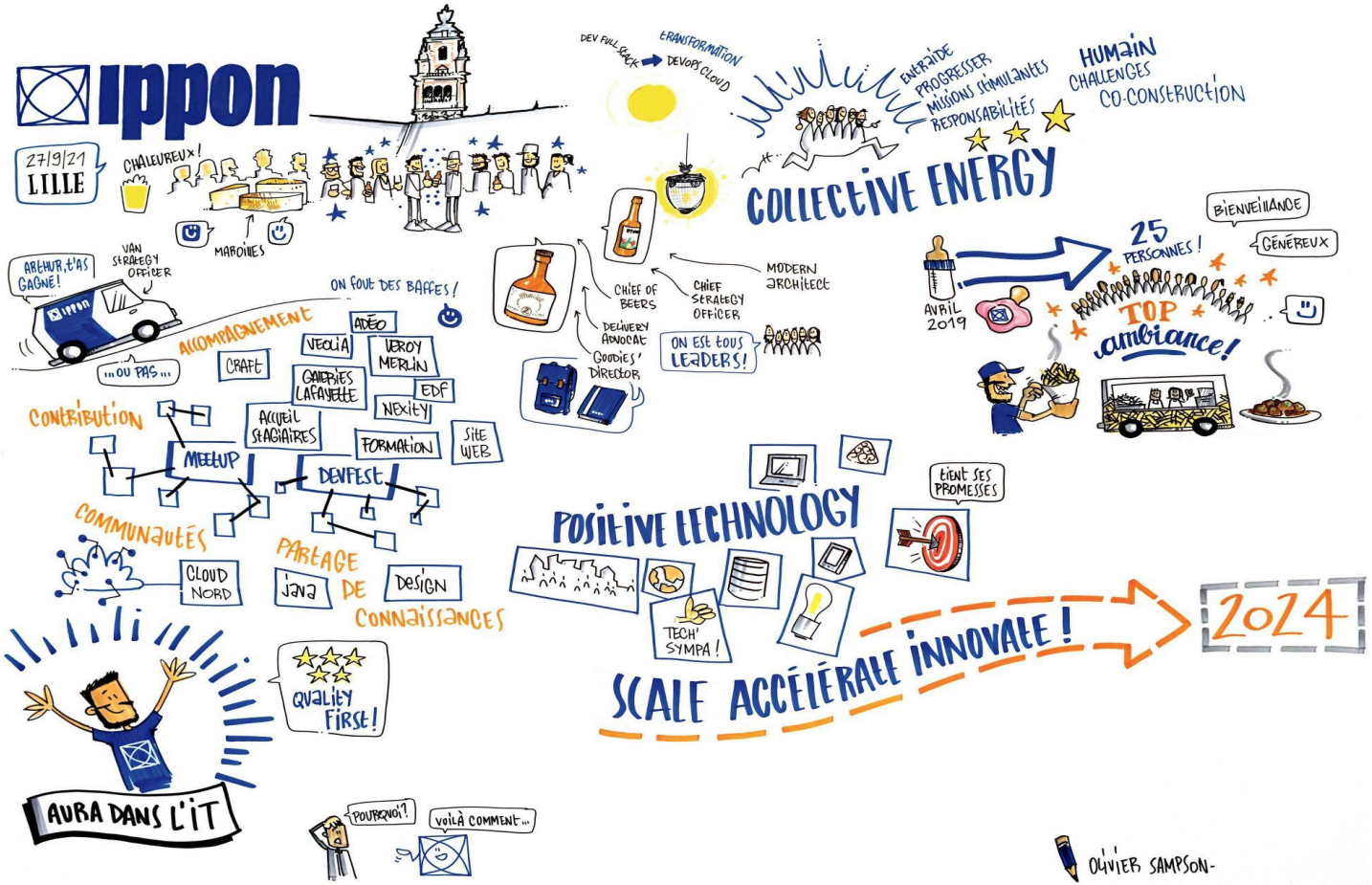
 # Enseignant vacataire depuis 2018 UPHF - DUT Première année

 # Joueur de Go, amateur de cinéma (un peu trop)

Sébastien JAUPART

https://github.com/sjaupart

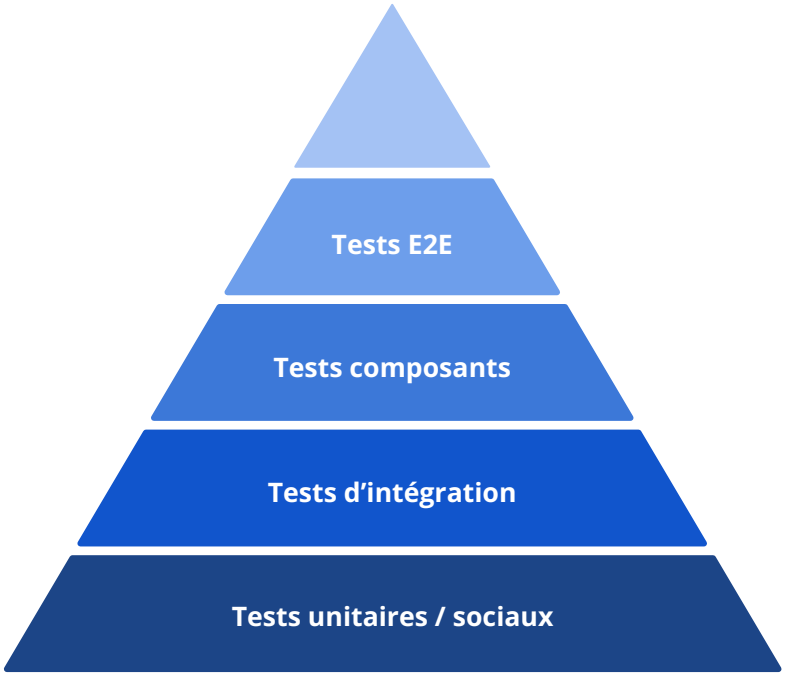
@s_jaupart

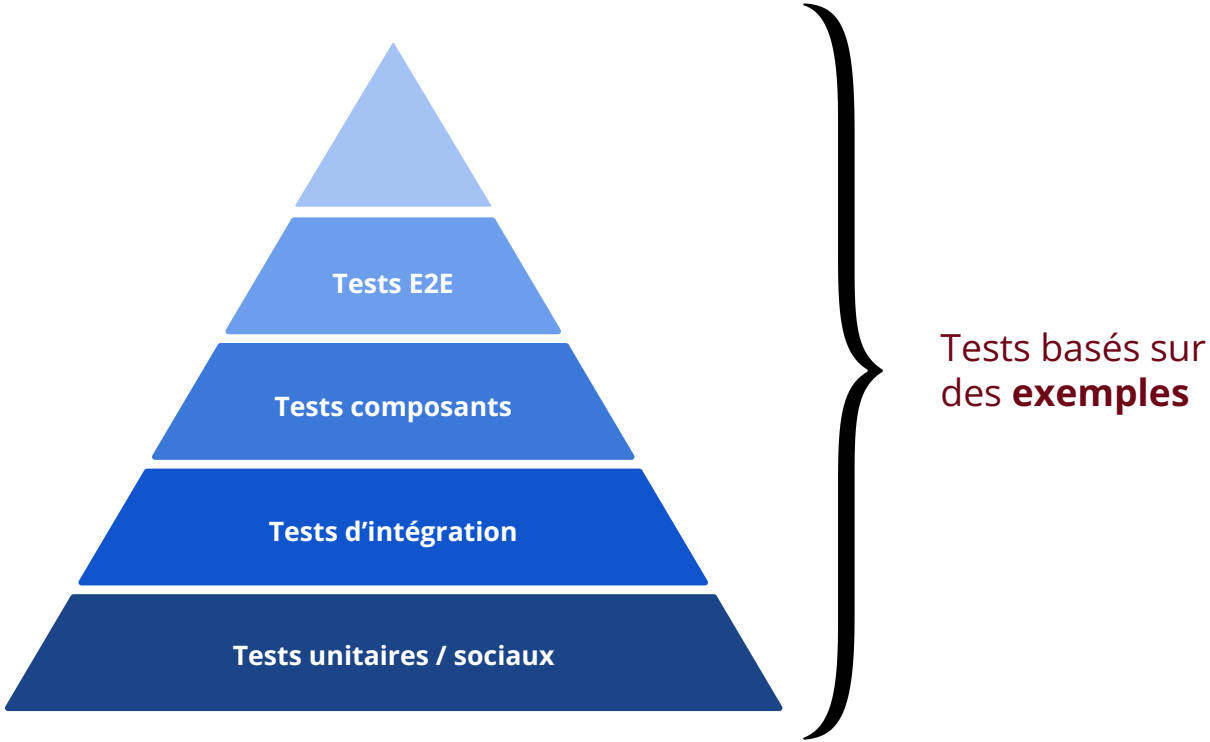

ippon
27/9/21
LILLE
CHALEUREUX!
MARQUES
DEV FULL-STACK
TRANSFORMATION
DEVOPS CLOUD
ENTRAIDE
PROGRESSER
MISSIONS STIMULANTES
RESPONSABILITÉS
HUMAIN
CHALLENGES
CO-CONSTRUCTION
COLLECTIVE ENERGY
ARTHUR, T'AS GAGNÉ!
VAN STRATEGY OFFICER
...OU PAS...
ON FOUT DES BAFFES!
ACCOMPAGNEMENT
ADEO
VECLIA
LEROY MERLIN
CRAFT
GALERIES LAFAYETTE
EDF
NEXITY
ACCUEIL STAGIAIRES
FORMATION
SITE WEB
CONTRIBUTION
MEETUP
DEVFEST
COMMUNAUTÉS
CLOUD NORD
PARTAGE DE CONNAISSANCES
JAVA
DESIGN
QUALITY FIRST!
AURA DANS L'IT
POURQUOI?
VOILÀ COMMENT...
CHIEF OF BEERS
CHIEF STRATEGY OFFICER
MODERN ARCHITECT
DELIVERY AVOCAT
GOODIES' DIRECTOR
ON EST TOUS LEADERS!
AVRIL 2019
25 PERSONNES!
BIENVEILLANCE
GÉNÉREUX
TOP ambiance!
POSITIVE TECHNOLOGY
TIENT SES PROMESSES
TECH' SYMPA!
SCALE ACCÉLÉRALE INNOVALE!
2024
OLIVIER SAMPSON-

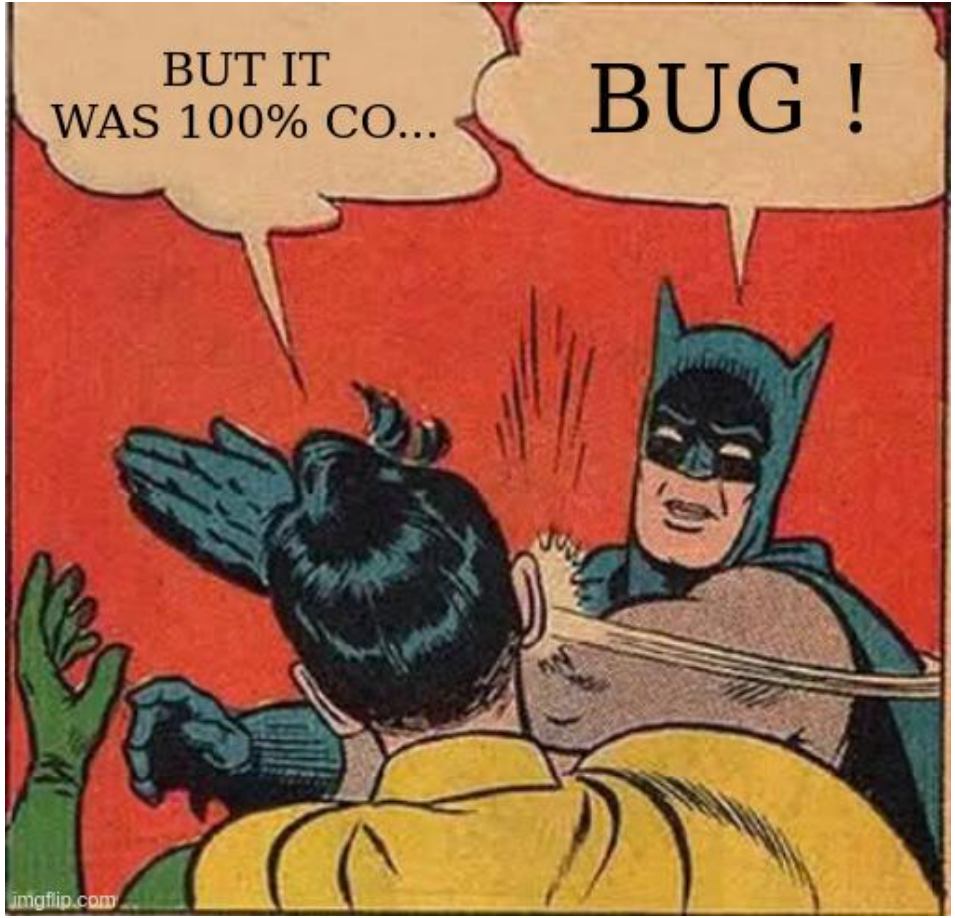
01 — **Mes tests unitaires ?**
Insuffisants ?

Photo by Dom Fou

Photo by Jason Goodman


Tests E2E
Tests composants
Tests d’intégration
Tests unitaires / sociaux


Tests E2E
Tests composants
Tests d'intégration
Tests unitaires / sociaux
Tests basés sur des **exemples**

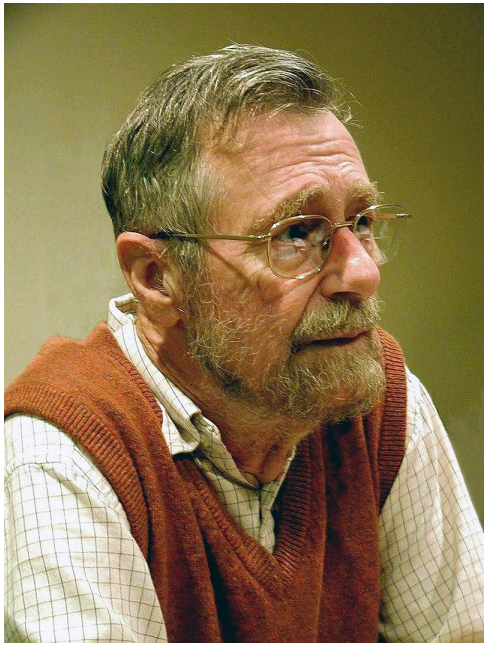

BUT IT
WAS 100% CO...
BUG !
imgflip.com


BUGS
LIGHTYEAR
BUGS EVERYWHERE
imgflip.com

Des limites aux exemples ?

*Quelles **contraintes liées aux exemples** m'empêchent de détecter ces anomalies ?*

Limite n°1 - La non-exhaustivité



❝ *Les tests prouvent la présence d'anomalies, mais* ***jamais leur absence*** ❞ *- Edsger W. Dijkstra*

Limite n°2 - Le souci du détail

Les informations inutiles occultent la véritable intention de nos tests.

```java
// Exemple Java / AssertJ

@Test
void price_value_must_be_positive() {
    assertThatThrownBy(() -> new Price( value: -1,  currency: "EUR"))
            .isInstanceOf(InvalidPriceException.class)
            .hasMessage("Price must have a positive value.");
}
```

Limite n°2 - Le souci du détail

```java
// Object Mother Pattern

@Test
void price_value_must_be_positive() {
    assertThatThrownBy(() -> Prices.withValue(-1))
            .isInstanceOf(InvalidPriceException.class)
            .hasMessage("Price must have a positive value.");
}

public class Prices {

    public static Price withValue(int value) {
        return new Price(value, currency: "EUR");
    }
}
```

Limite n°3 - L'explosion combinatoire

"Augmentation exponentielle du nombre de cas possibles par l'ajout d'un nouvelle donnée"

Limite n°3 - L'explosion combinatoire

"Augmentation exponentielle du nombre de cas possibles par l'ajout d'un nouvelle donnée"

Les besoins métier évoluent, les applications se complexifient.

Plus de cas possibles nécessitent plus de tests.

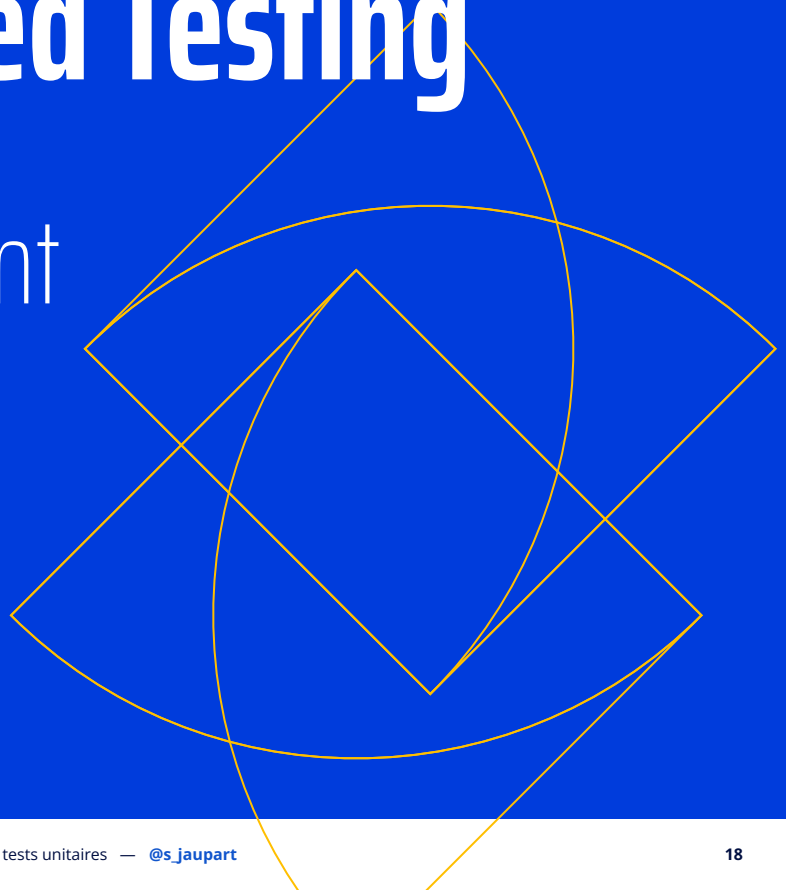
Que faire ?

Comment se rassurer davantage sur notre implémentation ?

- **Reproduire une situation “ISO prod”.**
- **Ignorer les détails, tester l’utile.**
- **Orienter nos tests sur le comportement plutôt que le résultat.**

02 — Property-based Testing

Une approche axée sur le comportement



Propriété, vous dites ?

*“Une propriété est une caractéristique d’un objet considérée comme étant **toujours vraie**.”*

Propriété, vous dites ?

Bowling - “Pour toute partie possible, le score doit se situer entre 0 et 300.”

Domaine du pricing - “Pour tout article distribué dans un pays interdisant la vente à perte, le prix de cet article est supérieur ou égal à son seuil de vente à perte.”

Pour toute valeur *appartenant à un ensemble* ***spécifique****,*

la propriété décrite doit être ***satisfaite****.*

Property-based Testing

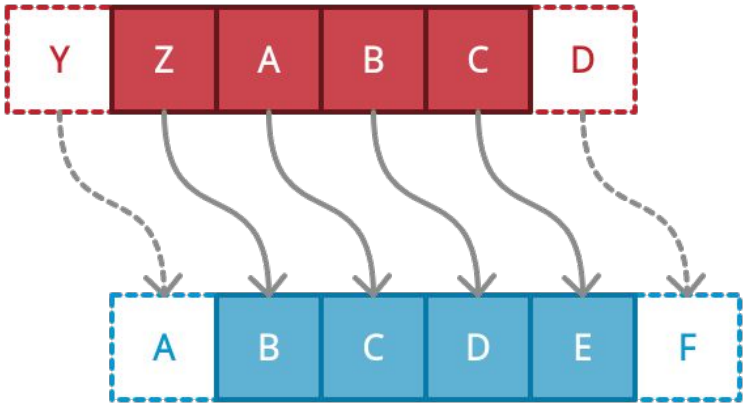
Popularisé par QuickCheck (Haskell)

L'objectif → Tester le **comportement** en validant les propriétés (ou "invariants").

La manière → S'appuyer sur la **génération aléatoire** de valeurs pour explorer et détecter les anomalies.

Property-based Testing… par l'exemple

Prenons l'exemple du chiffre de César.



Chiffrement avec une clé de 2

Property-based Testing… par l'exemple

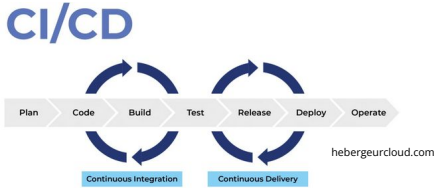
Time to demo !

Property-based Testing

Les tests de propriétés vivent à travers le temps.

Votre machine

Les machines des collègues



L'intégration continue

Mon test a (enfin) détecté un cas d'erreur

Comment reproduire un test en échec basé sur une génération aléatoire des valeurs d'entrée ?

Deux outils pour aider le développeur → la **seed** et la **réduction** (ou "*shrinking*").

Un test de propriété peut-il remplacer un test unitaire ?

*Dans **de rares cas**, c'est possible et suffisant.*

*… mais généralement, la réponse est **non**.*

La génération de données

Par défaut, des générateurs de données pour les types les plus basiques.

JUnit-QuickCheck :

- Types primitifs
- Strings
- Enumérations
- Collections standards (ArrayList, HashSet, HashMap, ...)

Pros/Cons

Avantages

- **Détection efficace des cas à la marge.**
- **Écriture concise et proche du besoin métier.**
- **Facilement reproductible.**
- **Les tests vivent à travers le temps.**

Inconvénients

- **“Test vert” ≠ “implémentation correcte”.**
- **Le feedback n’est pas immédiat.**
- **Un temps d’exécution allongé, mais négligeable.**

A la recherche des propriétés !

Propriétés classiques

- **Symétrie**
- **Idempotence**
- **Commutativité**
- **“Difficile à prouver, facile à vérifier”**
- **Test Oracle : pour le refactoring**
- **Et bien d’autres…**

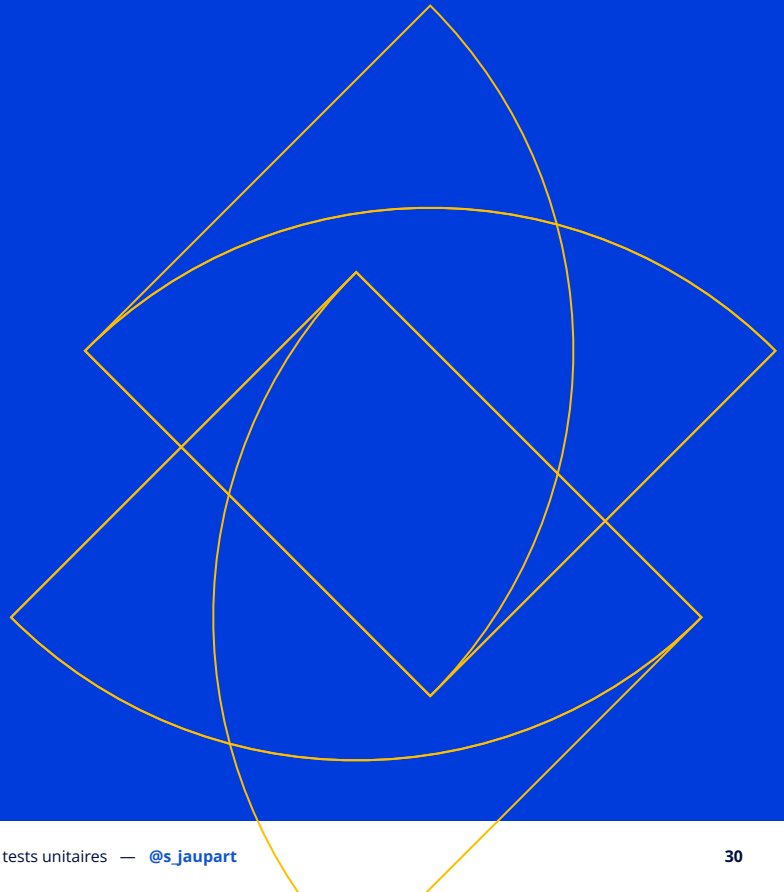
Propriétés liées aux règles métier

Toutes les règles métier possèdent des propriétés.

Le petit conseil pour débuter.

Rechercher parmi les tests existants si certains peuvent être facilement convertis / remplacés.

04 — En conclusion



— Merci !

Des questions ?

Twitter : s_jaupart

GitHub : sjaupart