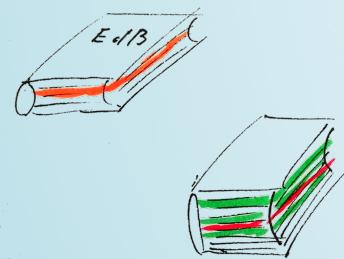


STRATÉGIES DE TESTS AGILE LA FAIRE BIEN POUR EN FAIRE MOINS

STRATÉGIES D'HIER

Le cahier de test est l'ensemble des tests manuel d'un système.

Avec le temps, il devient vite trop long à exécuter en entier avant chaque release.



Stratégie de tests :

L'art de sélectionner les tests fonctionnels (manuels) à jouer pour éviter les régressions et évidemment montrer la présence des évolutions

CONTENU

1. Stratégie de tests agile

- Qu'est-ce que c'est ?
- Comment on en fait une ?

2. Tactiques de tests

- Comment on décide de tester ou non

3. Techniques de tests

- Quelques tips pour se faciliter la vie

STRATÉGIES "AGILES"

Dans les "grands comptes", La stratégie de tests est devenu un document stable où l'équipe décrit ses engagements en matière de tests et contient :

STRATÉGIES "AGILES"

Dans les "grands comptes", La stratégie de tests est devenu un document stable où l'équipe décrit ses engagements en matière de tests et contient :

- L'application et son contexte

STRATÉGIES "AGILES"

Dans les "grands comptes", La stratégie de tests est devenu un document stable où l'équipe décrit ses engagements en matière de tests et contient :

- L'application et son contexte
- Ses enjeux et risques (promesses faites aux clients, nouvelle infra cloud, ...)

STRATÉGIES "AGILES"

Dans les "grands comptes", La stratégie de tests est devenu un document stable où l'équipe décrit ses engagements en matière de tests et contient :

- L'application et son contexte
- Ses enjeux et risques (promesses faites aux clients, nouvelle infra cloud, ...)
- Les types de tests et la quantité d'effort que l'on compte investir

APPLICATION ET CONTEXTE

La "bonne" stratégie de test est alignée sur la stratégie de l'entreprise comme le reste :

- La vision produit,
- L'infrastructure,
- L'architecture logicielle, ...

ENJEUX ET RISQUES

Pour ne pas en oublier, vous avez:

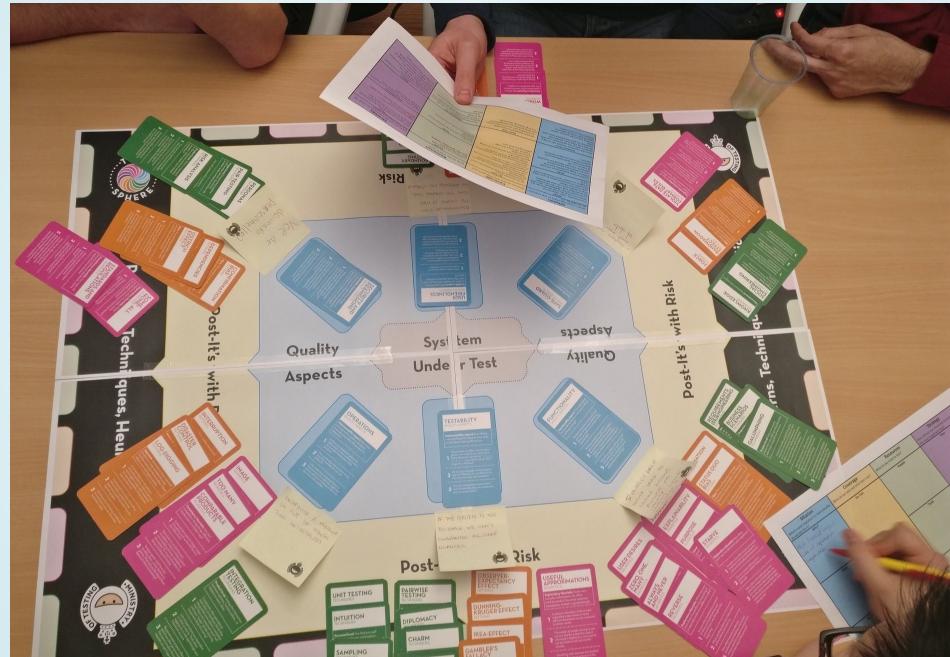


Figure 1. Les cartes de Test Sphere et son atelier, le Risk Storming

ENJEUX ET RISQUES

Pour ne pas en oublier, vous avez:



Figure 2. Deux listes dans la norme ISO 25010, en §4.1 et §4.2.

ENJEUX ET RISQUES

CoûtDuRisque =

*Probabilité(Incident) * ImpactFinancier(Incident)*

TESTS === FEEDBACKS

<i>Etape</i>	<i>Prescriptif</i>	<i>Descriptif</i>	<i>Observabilité</i>	<i>Démo</i>
Données d'entrées	Données de tests	Tests ou extrait de prod	Prod journalisée	Données de tests
Déclenchement	Test	Test	Prod journalisée	Démonstrateur
Système testé	Local	Local	Prod	Hors Prod
Données de sortie	Données capturées	Données capturées (et stabilisées)	Prod journalisée	Hors Prod
Vérifications	Assertions	Comparaisons avec sortie approuvée	Moniteur de journaux	Equipe

TESTS === FEEDBACKS

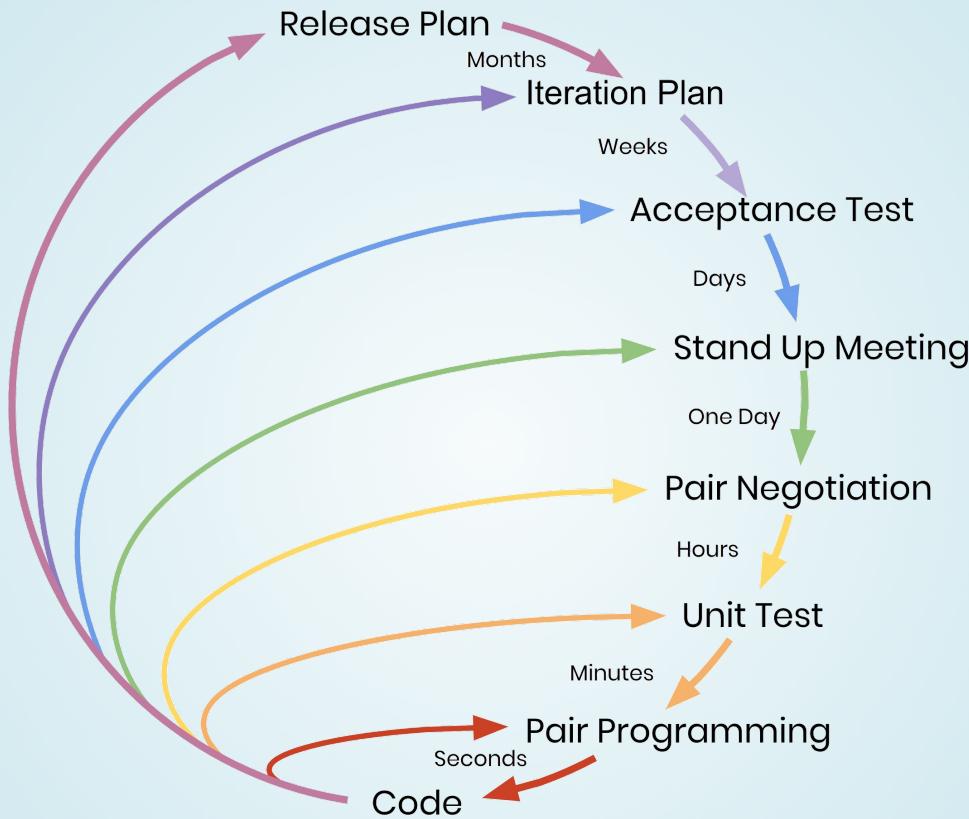


Figure 4. colorised XP feedback loops

Les boucles de feedback sont des tests sur l'équipe au lieu de tester le livrable.

TACTIQUES DE TESTS

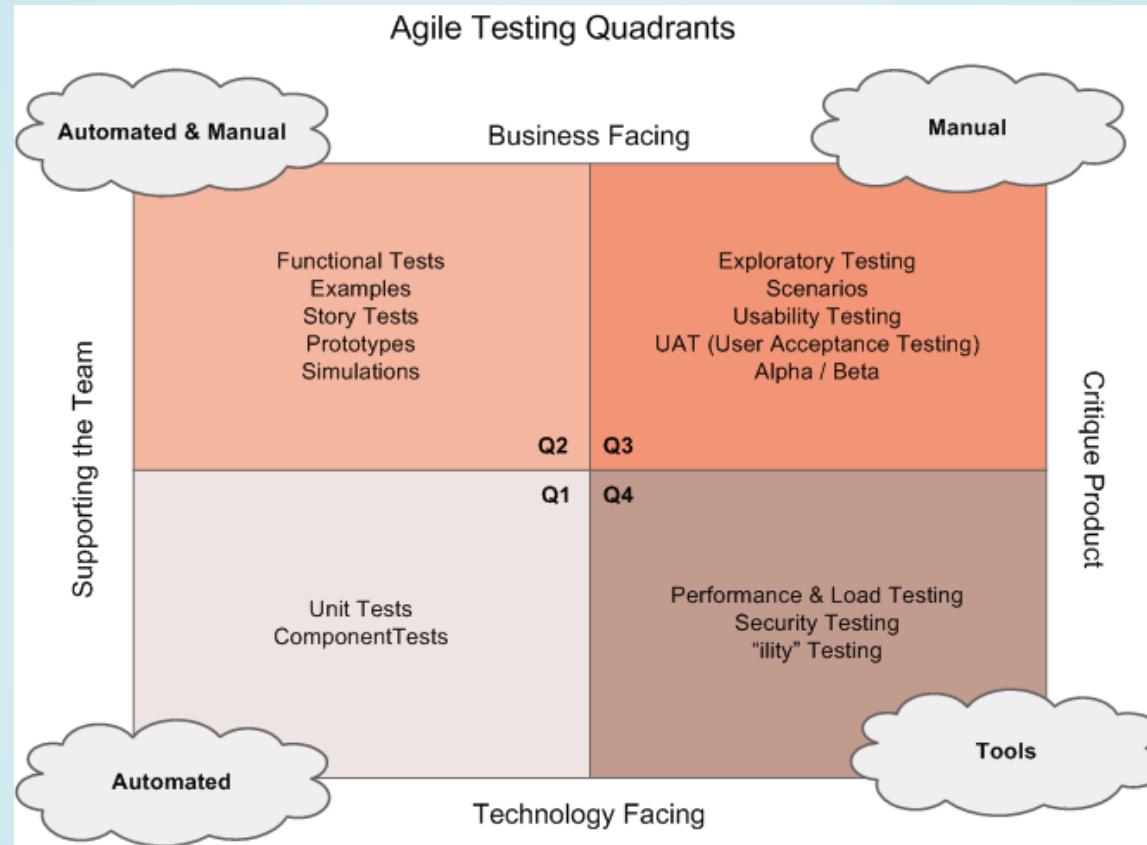


Figure 5. Quadrants de tests par Brian Marick et diffusé par Lisa Crispin

TACTIQUES DE TESTS

Les TUs de dev et les E2E de testeurs ne sont pas seuls.

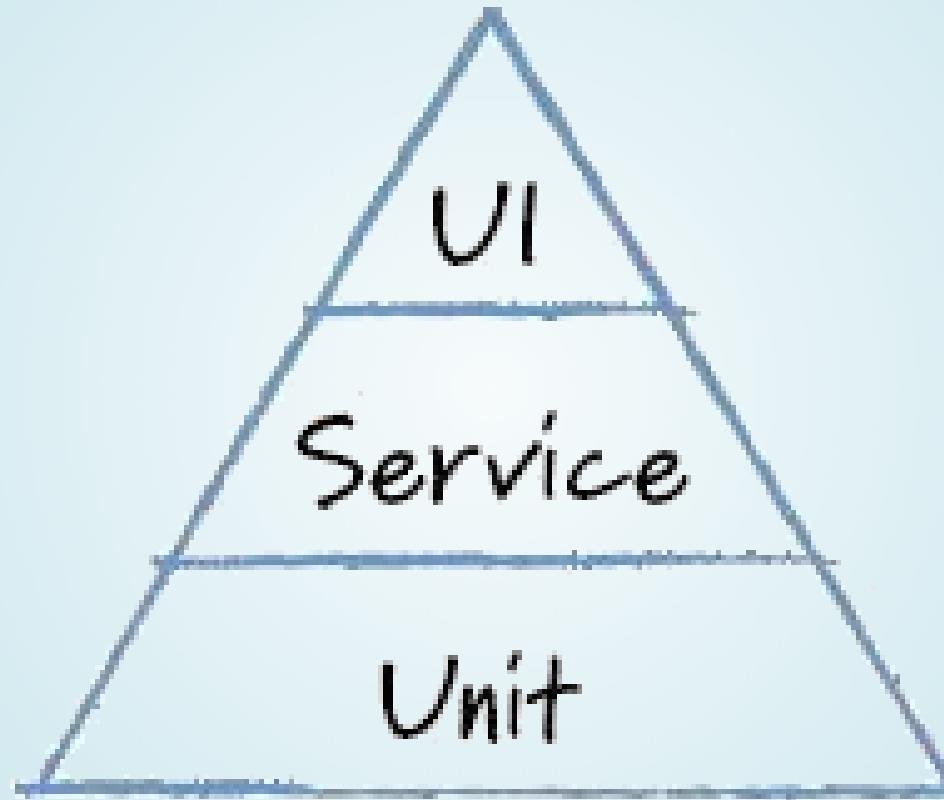


Figure 6. Mike Cohn introduit les TIs, "tests de composants"

TACTIQUES DE TESTS

Le bon niveau de tests pour le bon niveau d'abstraction

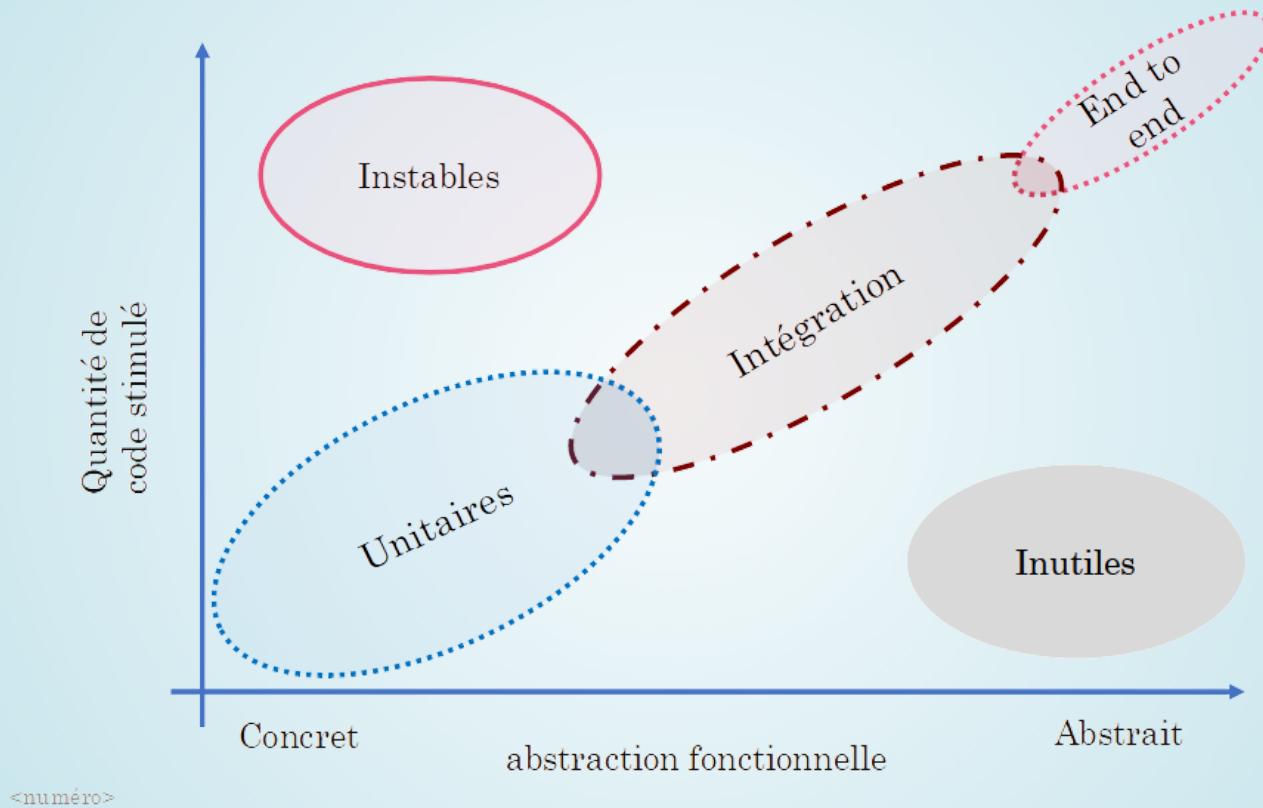


Figure 7. Gerard Meszaros, Find the Right Abstraction Level for Your Tests

TACTIQUES DE TESTS

En architecture hexagonale, c'est plus facile.

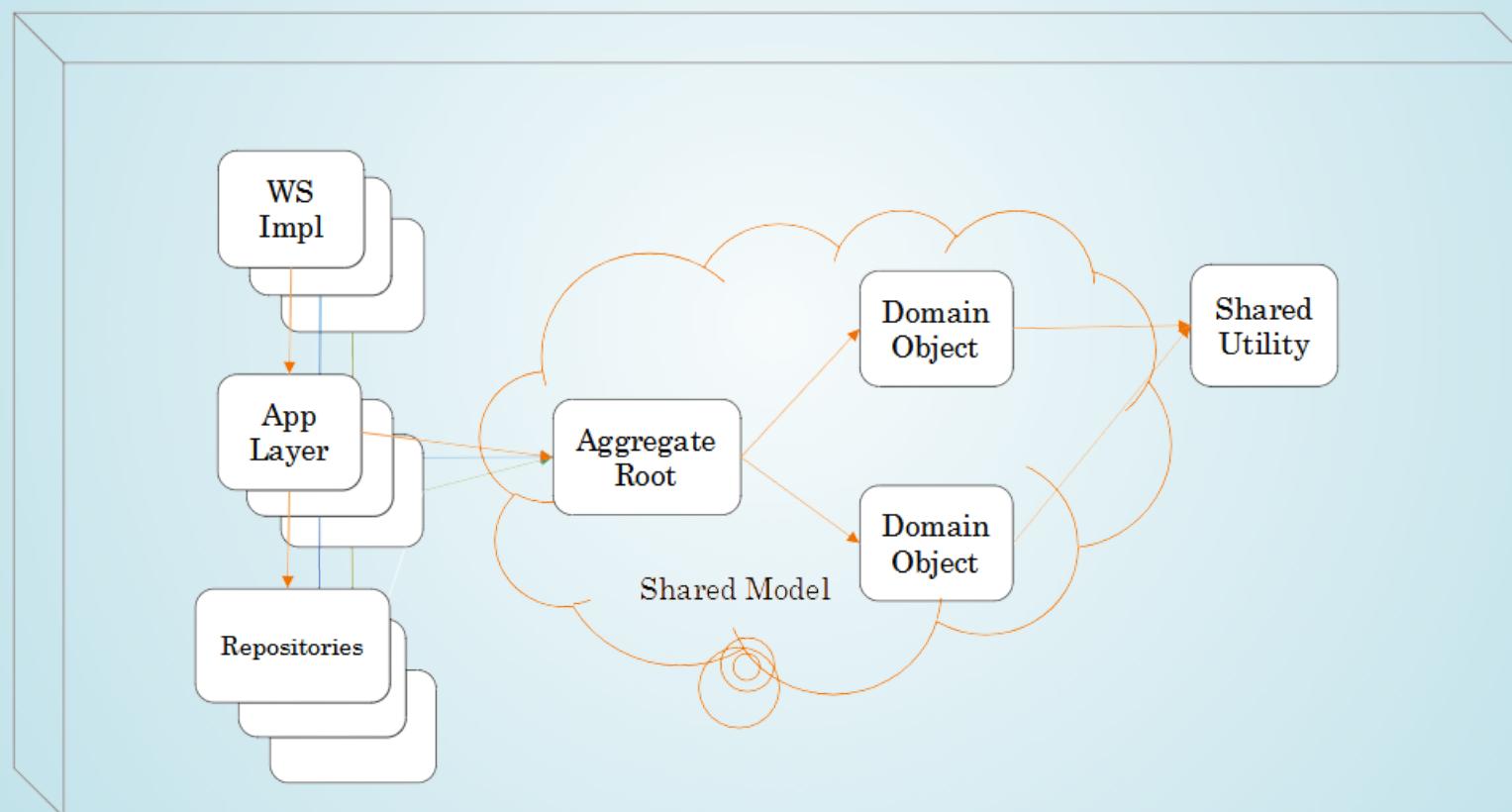


Figure 8. Une architecture testable

TACTIQUES DE TESTS

Pourquoi écrire ce test ?

TACTIQUES DE TESTS

Pourquoi écrire ce test ?

1. Probabilité de régression future sur ce code ?

TACTIQUES DE TESTS

Pourquoi écrire ce test ?

1. Probabilité de régression future sur ce code ?
 1. Du code dont on est propriétaire

TACTIQUES DE TESTS

Pourquoi écrire ce test ?

1. Probabilité de régression future sur ce code ?
 1. Du code dont on est propriétaire
 2. Algorithme avec des IFs, des ELSEs, WHILEs et/ou des FORs...

TACTIQUES DE TESTS

Pourquoi écrire ce test ?

1. Probabilité de régression future sur ce code ?
 1. Du code dont on est propriétaire
 2. Algorithme avec des IFs, des ELSEs, WHILEs et/ou des FORs...
 3. Pas de maîtrise des appelants futurs, règle de gestion transverse

TACTIQUES DE TESTS

Pourquoi écrire ce test ?

1. Probabilité de régression future sur ce code ?
 1. Du code dont on est propriétaire
 2. Algorithme avec des IFs, des ELSEs, WHILEs et/ou des FORs...
 3. Pas de maîtrise des appelants futurs, règle de gestion transverse
2. Documenter/Justifier ce code par un exemple concret pour la future équipe ?

TECHNIQUES DE TESTS

- Données d'entrées
- Types de vérifications
- Types de tests

DONNÉES D'ENTRÉES

```
List<Product> catalog = Given.theStandardCatalog();
Customer simpleClient = Given.theFreemiumCustomer();
Customer goldClient = Given.theSimpleGoldenCustomer();
Customer specificGoldClient = Given.aGoldenCustomer()
    .withLoyaltyPoints(200)
    .withPurchaseHistory(
        Given.anInvoice(
            Given.product("productName", /* price in cents */ 12300),
            Given.product("productName", 12300),
            Given.product("anotherProductName", 1200)
        )
    ).build();
/* specificGoldClient = */ fromClientRepository(specificGoldClient)
```

DONNÉES D'ENTRÉES

```
public class Given {  
    public static CustomerBuilder aGoldenCustomer() {  
        return new CustomerBuilder().from(Given.theSimpleGoldenCustomer());  
    }  
    public static class CustomerBuilder {  
        private String firstName; [...]  
        CustomerBuilder from(Customer source) {  
            return this.withCivility(source.getFisrtName(), source.getLastName());  
        }  
        CustomerBuilder withLoyaltyPoints(int points) [...]  
        CustomerBuilder withPurchaseHistory(InvoiceBuilder... invoice)  
    }  
}
```

TESTS PAR ASSERTIONS

```
ReservationDTO reservation =  
whenMakeReservationFor(  
    new ReservationRequestDTO(trainId, numberOfSeats));  
  
assertThat(reservation)  
.isEqualToComparingFieldByField(  
    new ReservationDTO(trainId, booking, asList("1A", "2A")));
```

Les librairies d'assertions adaptent les vérifications au type de données à vérifier.

Les objets/json, les collections sont plus faciles à tester.

Les assertions "maison" permettent de faire des vérifications "métiers".

Certaines données contiennent des valeurs premières et des valeurs calculées.

Les vérifications sur les valeurs premières seules suffisent.

TESTS PAR APPROBATIONS

```
[... usual test ...]
String output = captureOutputs();
/* dates and PKs change with different executions */
String outputNormalised = replaceMovingData(output);
/* verify diff received output against a previously approved output */
Approvals.verify(outputNormalised);
```

Parfois, on ne sait pas à l'avance quel est le résultat attendu.

Cette démarche s'appelle "Approval Testing".

Elle est utilisée sur du code qui est déjà en production.

Elle est aussi utile lorsque le développeur n'est pas autonome pour valider la sortie
(export comptable, génération de document légal, etc.)

AUTRES TYPES DE TESTS

Pour relever le niveau de confiance :

- Les tests paramétrés permettent le "property based testing"
- Le "mutation testing" challenge vos tests unitaires

Quand les données d'entrées changent en prod
sans prévenir,

Le code de prod est designé comme un code de test sur les données.

Pour les questions :

- Sur place
- Sur twitter :
@CoulasFedou
- sur Linked-In :
nicolas-fedou

Pour les feedbacks :



<https://roti.express/r/alp-14>