Tests manuels ou tests automatisés ?

Les tests visent à vérifier que le produit codé fonctionne comme prévu selon des scénarios prédéfinis et représentatifs.

Modalités de test

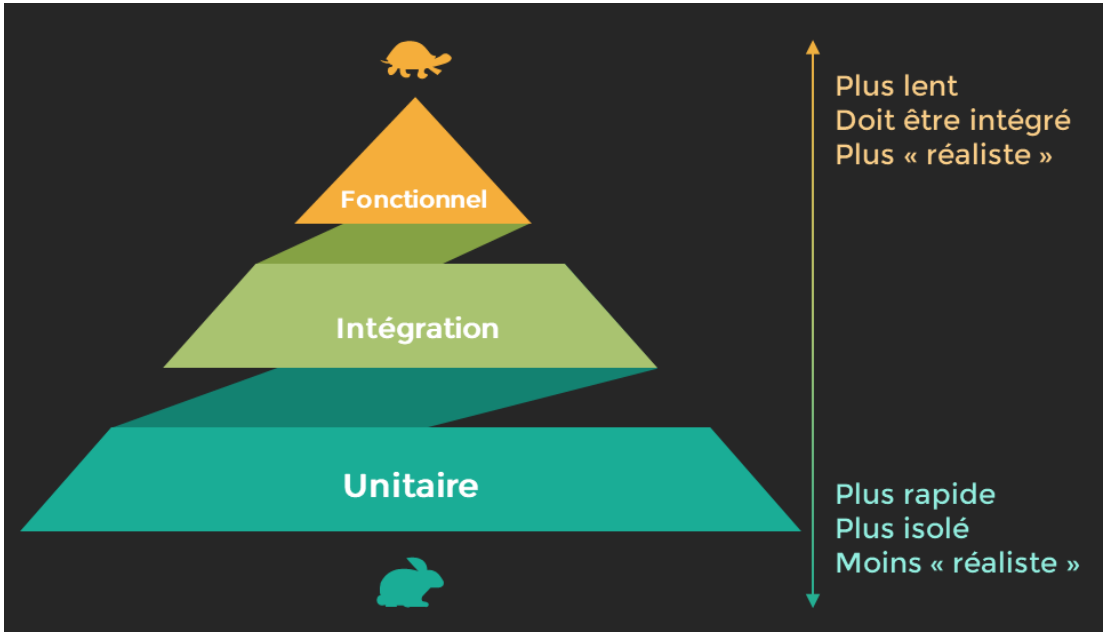
- Test manuel : test "par l'humain" avec revue de code, de spécifications, de documents, ...
- Test automatisé : test par l'exécution du système pour s'assurer d'un fonctionnement correct

Actuellement, le test automatisé est la méthode la plus utilisée et représente jusqu'à 60% de l'effort complet de développement d'un produit logiciel.

Définitions

- *Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts* - G. Myers (The Art of Software testing)
- *Le test est l'exécution ou l'évaluation d'un système ou d'un composant par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus* - IEEE
  $\rightarrow$ notion d'Oracle : résultats attendus d'une exécution du logiciel
- *Testing can reveal the presence of errors but never their absence* – Edsgar W. Dijkstra. *Notes on structured programming*. Academic Press, 1972.
  $\rightarrow$ test exhaustif impossible à réaliser, le test est une méthode de vérification partielle

Les différents types de tests automatisés

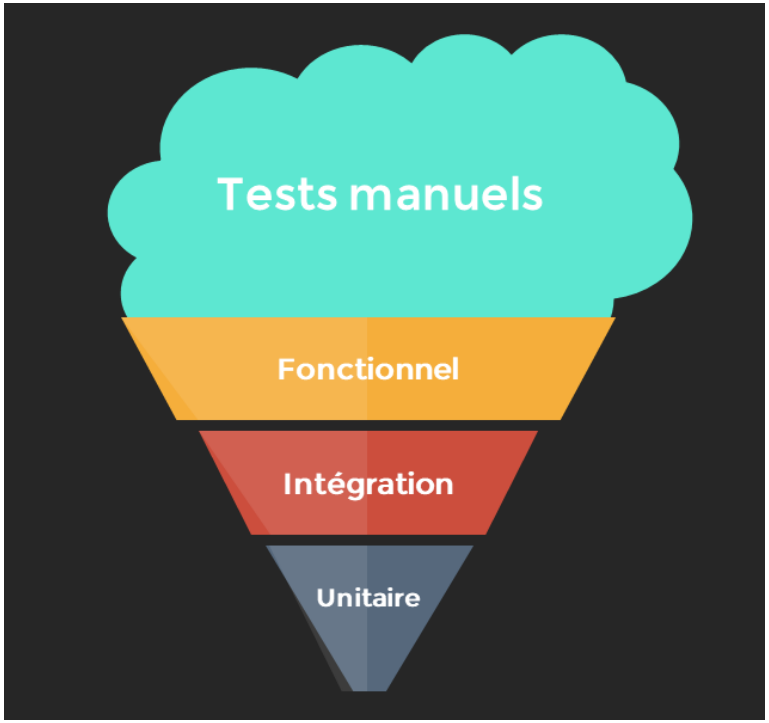


Pyramide des tests, *Succeeding with Agile*, Mike Cohn.

3 types de tests automatisés les plus courants, répartis dans un projet de développement agile selon la pyramide ci-dessus.

- tests unitaires : vérification de "petites" unités de code (méthodes) en isolation.
- tests d'intégration : vérifient si les unités de code fonctionnent ensemble comme prévu
- tests fonctionnels : simuler le comportement d'un utilisateur final sur l'application. Code développé considéré comme une boîte noire.

Les mauvaises pratiques



Anti-pattern de la pyramide des tests, *Succeeding with Agile*, Mike Cohn.

- Plus un bug est détecté tôt, moins son coût de correction est élevé
- L'automatisation des tests n'a de réelle valeur ajoutée que si l'on arrive à exécuter ces tests sans les modifier de nombreuses fois (plus facile de rentabiliser un test unitaire qu'un test fonctionnel)

Test unitaire

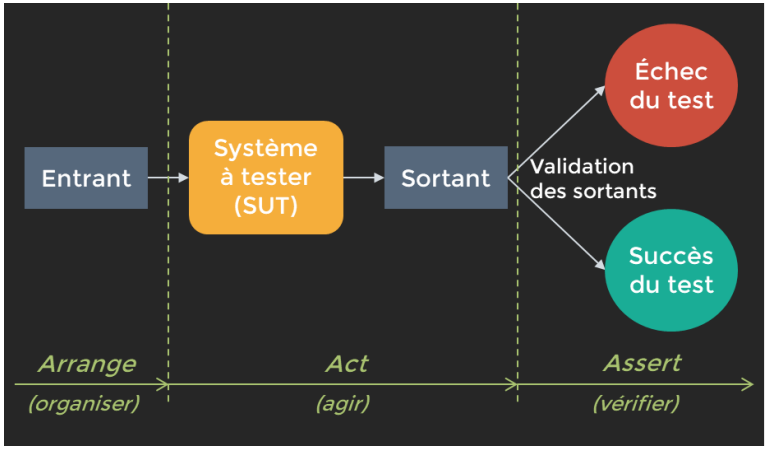
3 règles

- tester le plus possible : afin d'augmenter les chances de découvrir des bugs
- tester le plus tôt possible : plus les tests sont fait tôt plus les bugs sont rapidement détectés
- tester le souvent possible : en les automatisant et si possible en les intégrant dans un processus d'intégration continue

Principes des Tests unitaire

- le test doit être le plus petit et le plus simple possible
- les TU doivent être automatisés

Méthode de test unitaire



Le système à tester donne des résultats (ou sortants) à partir de données de test (ou entrants).

Structuration d'un cas de test unitaire

Selon la méthode AAA (Arrange - Act - Assert) ou *Given - When -Then* (issu du BDD Behaviour-driven development) :

- Given : initialiser tous les entrants ou Données de Test et le système à tester si besoin.
- When : exécutez le système à tester avec les entrants précédemment initialisés dans des sortants que vous conservez.
- Then : vérifiez les sortants en fonction de ce qui est attendu par rapport à vos entrants (oracle). Vous en concluez alors si c'est en succès ou en échec.

Un oracle permet de décider la réussite d'un scénario de test.

Méthode de test unitaire : Exemple

```java
import Calculatrice.Calculatrice;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class TestCalculette {
    @Test
    public void testAjouter(){
        //Given: entrants / donnees d'entrees a tester
        Calculatrice calc = new Calculatrice( a: 4, b: 5);

        //When: execution avec entrants, resultat mis dans sortant
        int resultatEffectif = calc.ajouter();

        //Then: verification des sortants en fonction de ce qui est attendu (oracle)
        int resultatOracle = 9;
        assertEquals(resultatOracle,resultatEffectif);


    }

}
```

```java
package Calculatrice;

public class Calculatrice {
    private int x;
    private int y;

    public Calculatrice(int a,int b){
        x = a;
        y = b;
    }

    public int ajouter(){
        x -= y;
        return x;
    }


}
```

Comment choisir les données de tests (DT) ?

- test exhaustif impossible à réaliser
- les DT ou entrants doivent permettre d'avoir un échantillon représentatif de toutes les entrées possibles de la méthode à tester

→ comment choisir les données de tests ?

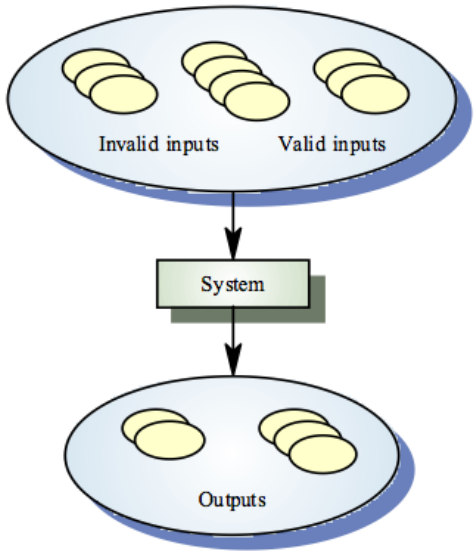
1. Approche aléatoire
2. Analyse partitionnelle des domaines des données d'entrée
3. Tests aux limites

1. Test aléatoire

- Principe : utiliser une fonction aléatoire pour sélectionner les DT sur leurs domaines de définition
- Facilement automatisable mais besoin d'un oracle valable pour tous les cas de tests
- Difficultés à produire des comportements très spécifiques
- Aucune garantie de bonne couverture de l'ensemble des entrées
- Référentiel pour les autres techniques

2. Analyse partitionnelle

- L'objectif est de diminuer le nombre de cas de tests en calculant des classes d'équivalence
- Une classe d'équivalence correspond à un ensemble de données de tests supposées tester le même comportement, i.e. activer le même défaut.
- Principe : Partitionner le domaine d'entrée de la méthode à tester en un nombre fini de classes d'équivalence.



2. Analyse partitionnelle

Trois phases :

1. Pour chaque donnée d'entrée de la méthode à tester, définir des classes d'équivalence valides et invalides sur les entrées

   - Si la donnée d'entrée est un intervalle, construire une classe d'équivalence valide (valeur dans intervalle) et 2 classes invalides (valeur inf / sup)
   - Si la donnée est un ensemble de valeurs, construire une classe valide avec valeurs dans l'ensemble, une classe avec ensemble vide, une classe invalide avec trop de valeurs
   - Si la donnée est une obligation ou une contrainte, construire une classe valide avec la contrainte respectée, une classe invalide avec la contrainte non-respectée

2. Analyse partitionnelle

Exemple :

Tester un programme calculant la valeur absolue d'un entier à partir d'une entrée au clavier (donc une chaîne de caractères) supposée fournie en notation décimale.

classe	validité
chaîne vide	invalide
plusieurs mots	invalide
pas un décimal	invalide
décimal positif	valide
décimal négatif	valide

2. Analyse partitionnelle

1. Pour chaque donnée d'entrée, définir des classes d'équivalence valides et invalides sur les entrées
2. Définir un oracle et au moins un représentant par classe d'équivalence,

Exemple

Tester un programme calculant la valeur absolue d'un entier à partir d'une entrée au clavier (donc une chaîne de caractères) supposée fournie en notation décimale.

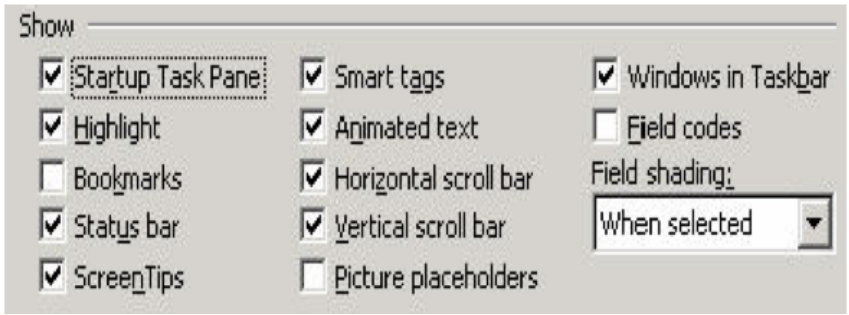
classe	validité	représentant	oracle
chaîne vide	invalide	""	échec
plusieurs mots	invalide	"1234 1234"	échec
pas un décimal	invalide	"56a"	échec
décimal positif	valide	"1234"	1234
décimal négatif	valide	"-1234"	1234

2. Analyse partitionnelle

1. Pour chaque donnée d'entrée, définir des classes d'équivalence valides et invalides sur les entrées
2. Définir un oracle et au moins un représentant par classe d'équivalence,
3. Choisir les DT pour chaque cas de test :
   - technique *all single* : générer des jeux de tests utilisant au moins une fois un représentant de chaque classe d'équivalence de chaque paramètre
   - technique *all pairs* : générer des jeux de tests utilisant au moins une fois chaque paire de représentants de classe d'équivalence

2. Analyse partitionnelle et technique *all pairs*

- Les combinaisons de valeurs de domaines d'entrée donnent lieu à une explosion combinatoire
- Exemple : Options d'une boite de dialogue



$\rightarrow 2^{12} \times 3 = 12288$

- Tester un fragment des combinaisons de valeurs qui garantissent que chaque combinaison de 2 variables est testée
- L'idée sous-jacente : la majorité des fautes sont détectées par des combinaisons de 2 valeurs de variables

2. Analyse partitionnelle et technique *all pairs*

Exemple

4 variables avec 3 valeurs possibles chacune

OS	Réseau	Imprimante	Application
XP	IP	HP35	Word
Linux	Wifi	Canon900	Excel
Mac X	ATM	Canon-EX	Pwpoint

81 combinaisons, 9 paires

	OS	Réseau	Imprimante	Application
Case 1	XP	ATM	Canon-EX	Pwpoint
Case 2	Mac X	IP	HP35	Pwpoint
Case 3	Mac X	Wifi	Canon-EX	Word
Case 4	XP	IP	Canon900	Word
Case 5	XP	Wifi	HP35	Excel
Case 6	Linux	ATM	HP35	Word
Case 7	Linux	IP	Canon-EX	Excel
Case 8	Mac X	ATM	Canon900	Excel
Case 9	Linux	Wifi	Canon900	Pwpoint

Exercice : Analyse partitionnelle

Proposez une analyse partitionnelle du domaine pour les méthodes suivantes (définir les classes valides, proposer des DT) :

1. Une méthode retourne le maximum entre deux nombres.
2. Une méthode retourne vrai si on entre un nombre pair et faux sinon.
3. Une fonction calculant la valeur absolue d'un entier.
4. Une fonction calculant la distance entre deux points du plan $(d((x, y), (x', y')) = \sqrt{(x - x')^2 + (y - y')^2})$.

Exercice : Analyse partitionnelle (correction)

Proposez une analyse partitionnelle du domaine pour les méthodes suivantes (définir les classes valides, proposer des DT) :

1. Une méthode retourne le maximum entre deux nombres. 3 classes d'équivalence :

$$C1(a,b) = \{a > b\}; C2(a,b) = \{a < b\}; C3(a,b) = \{a == b\}$$

$$DT = (5,2), (2,5), (5,5)$$

2. Une méthode retourne vrai si on entre un nombre pair et faux sinon.
3. Une fonction calculant la valeur absolue d'un entier.
4. Une fonction calculant la distance entre deux points du plan $(d((x,y),(x',y')) = \sqrt{(x-x')^2 + (y-y')^2})$.

Exercice : Analyse partitionnelle (correction)

Proposez une analyse partitionnelle du domaine pour les méthodes suivantes (définir les classes valides, proposer des DT) :

1. Une méthode retourne le maximum entre deux nombres.
2. Une méthode retourne vrai si on entre un nombre pair et faux sinon. 2 classes d'équivalence :

$$C1(d) = \{d\%2 == 0\}, C2(d) = \{d\%2 == 1\}$$

$$DT = (2), (3)$$

3. Une fonction calculant la valeur absolue d'un entier.
4. Une fonction calculant la distance entre deux points du plan ($d((x, y), (x', y')) = \sqrt{(x - x')^2 + (y - y')^2}$).

Exercice : Analyse partitionnelle (correction)

Proposez une analyse partitionnelle du domaine pour les méthodes suivantes (définir les classes valides, proposer des DT) :

1. Une méthode retourne le maximum entre deux nombres.
2. Une méthode retourne vrai si on entre un nombre pair et faux sinon.
3. Une fonction calculant la valeur absolue d'un entier. 3 classes d'équivalence :

$$C1(d) = \{d < 0\}, C2(d) = \{d > 0\}, C3(d) = \{d == 0\}$$

4. Une fonction calculant la distance entre deux points du plan ($d((x, y), (x', y')) = \sqrt{(x - x')^2 + (y - y')^2}$).

Exercice : Analyse partitionnelle (correction)

Proposez une analyse partitionnelle du domaine pour les méthodes suivantes (définir les classes valides, proposer des DT) :

1. Une fonction calculant la distance entre deux points du plan $(d((x,y),(x',y')) = \sqrt{(x-x')^2 + (y-y')^2})$.
   1. Considérer la relation entre x et x' ($<$, $=$ ou $>$: 3 classes d'équivalences pour x) et celle entre y et y' (3 classes d'équivalences pour y) : partitionnement en $3^2 = 9$ classes représentant les combinaisons possibles.
   2. Considérer le signe de x, x', y et y' (< 0, $= 0$ ou > 0) : partitionnement complet donne $3^4 = 81$ classes.
   3. Combinaison des deux partitionnements : 9×81 combinaisons (un certain nombre impossibles i.e. $x < x'$, $x = 0$ et $x' < 0$).

3. Tests aux limites

Solliciter les entrées se trouvant aux frontières des classes d'équivalence.

1. Calculer les classes d'équivalences
2. Pour chacune, générer une valeur médiane et une ou plusieurs valeurs aux bornes

   - Pour un domaine de type intervalle : on garde les 2 valeurs des limites, et les 4 valeurs pour les limites + / - le plus petit delta possible
   - Pour un ensemble ordonné de valeurs, on choisit le premier, le second, l'avant dernier et le dernier
   - Si une condition d'entrée spécifie un nombre de valeurs, définir les cas de test à partir du nombre minimum et maximum de valeurs, et des tests pour des nombres de valeurs hors limites invalides.

Le test aux limites produit à la fois des cas de test nominaux et de robustesse

3. Tests aux limites : exemple

Solliciter les entrées se trouvant aux frontières des classes d'équivalence.

1. Calculer les classes d'équivalences
2. Pour chacune, générer une valeur médiane et une ou plusieurs valeurs aux bornes

Test aux limites d'une fonction qui attend "oui" ou "non"

classe	validité	représentant avec limites
"oui"	valide	"oui"
"non"	valide	"non"
autre	invalide	" " ou "hello" ou "ouii"

3. Tests aux limites : exemple

Solliciter les entrées se trouvant aux frontières des classes d'équivalence.

1. Calculer les classes d'équivalences
2. Pour chacune, générer une valeur médiane et une ou plusieurs valeurs aux bornes

Test aux limites d'une fonction manipulant des numéros de département

classe	validité	représentant avec limites	Large couverture
$[1;95]$	valide	1,48,95	1,2,48,94,95
$[minInt;1[$	invalide	-3000,0	-3000,-1,0
$]95;maxInt]$	invalide	96,1000	96,97,1000

Exercice : Tests aux limites

Proposez des tests aux limites pour une fonction qui incrémente un entier.

Exercice : Tests aux limites

Proposez des tests aux limites pour une fonction qui incrémente un entier.

- entier minimal représentable $-intMax$,
- entier $-intMax + 1$
- très grands entiers $intMax - 1$ et $intMax$
- -1, 0 et $+1$.

Outils du test unitaire automatique

Framework d'automatisation des TU

Ecriture et automatisation des TU :

- Java : JUnit, TestNG
- PHP : PHPUnit, atoum, SimpleTest
- C++ : CppUnit
- .NET : NUnit

Outils d'analyse de couverture de code

Mesure de la couverture de code d'un jeu de tests :

- Java : Cobertura, Emma
- PHP : Xdebug

Outils d'automatisation des tests IHM

- Selenium pour les applications web

Outils du test unitaire automatique

JUnit + Eclemma