IUT Nantes
Pôle Sciences et technologie

Nantes Université

Conception de tests fonctionnels et implémentation de tests de levée d'exception

Jean-Marie Mottu (Lanoix, Le Traon, Baudry, Sunye)

IUT Nantes
Pôle Sciences et technologie

Nantes Université

Implémentation des tests de levées d'exceptions

Les exceptions

- Mécanisme permettant d'écrire des tests vérifiant la levée des exceptions
  - La levée des exceptions est positive : elle protège
  - Si un bug empêche la levée d'une exception, le programme risque
    - de planter,
    - de renvoyer de mauvais résultats.

- Le test des levées d'exception change à chaque version de Junit : ce sont des tests importants mais pas simples.

Test d'une méthode levant une exception

```
/**
 * Age humain d'un chien
 * @param theDog : Chien
 * @return ageHumain : flottant
 * @throws ArithmeticException
 */
fun ageHumain(theDog: Chien): int {
    if (theDog.age <= 0) {
        throw ArithmeticException("Un age est positif ou nul")
    }else {
        return theDog.age/12*7 //potentiellement faux
    }
}
```

La version basique
Calquée le test d'exception sur leur utilisation

```
@Test
fun testAgeHumainFail(){
    var lassie = Chien(nom:"Lassie", race:"yorkshire", mois: -4)
    try {
        chenil. ageHumain(lassie) //instancié précédemment
        //executed if exception not raised => test fails
        fail()
    }catch (e: ArithmeticException){
        //executed if exception raised as expected => test passes
    }
}
```

La version Junit 5 utilisant une assertion

```
@Test
fun testAgeHumainAssert(){
    var lassie = Chien(nom:"Lassie", race:"yorkshire", mois: -4)
    assertThrows<ArithmeticException> {
            chenil. ageHumain(lassie) //instancié précédemment
    }
}
```

Attention aux mauvaises levées d'exception

Si votre test vérifie la bonne levée d'une exception particulière, il ne faut pas qu'une autre soit lancée à la place.

```
@Test
fun testAgeHumainAutreExcept(){
   var lassie = Chien(nom:"Lassie", race:"yorkshire", mois: -4)
   assertThrows<NomInapproprieException> {
         chenil. ageHumain(lassie) //instancié précédemment
   }
}
```

Attention à ne pas attendre de levées d'exception à tort

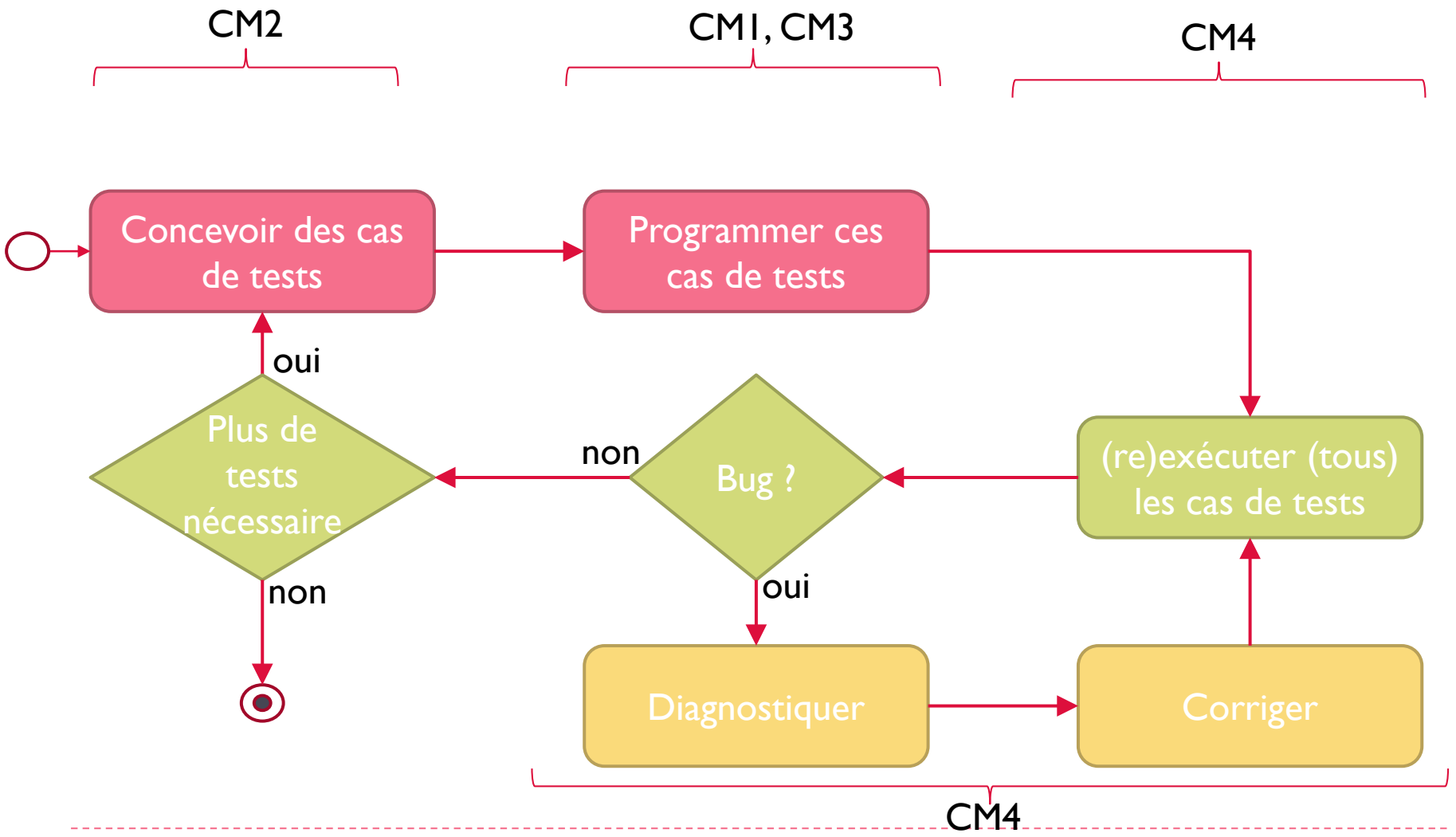
```
@Test
fun ageHumainFailBis(){
   var lassie = Chien(nom:"Lassie", race:"yorkshire", mois:24)
   try {
      chenil. ageHumain(lassie) //instancié précédemment
      //executed if exception not raised => test fails
      fail()
   }catch (e: ArithmeticException){
      //executed if exception raised as expected => test passes
   }
}
@Test
fun ageHumainCorrect(){
   var lassie = Chien(nom:"Lassie", race:"yorkshire", mois:24)
   assertEquals(14, chenil. ageHumain(lassie))
}
```

IUT Nantes
Pôle Sciences et technologie
Nantes Université

Test fonctionnel

Jean-Marie Mottu (Lanoix, Sunye, Le Traon, Baudry)

Le cycle de test dynamique



Problème d'exhaustivité : Cas de test pour l'addition

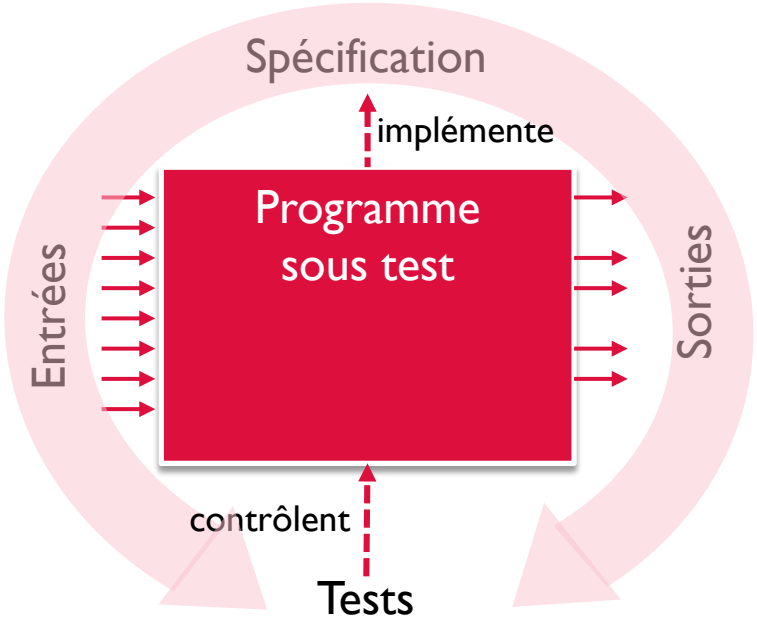
- Test de l'addition x + y = z
- Cas de test 1
  - Description du cas de test : vérifier que l'addition de 2 nombres opposés donne 0
  - Initialisation : allumer le programme, etc.
  - Donnée de test : (x = 5 , y = -5)
  - Oracle : (res == 0)
- Cas de test 2
  - Description du cas de test : vérifier que l'addition de 2 fois le même nombre donne son double
  - Initialisation : allumer le programme, etc.
  - Donnée de test : (x = 2 , y = 2)
  - Oracle : (res == 4) ou (res == 2* x)
- etc. jusqu'où/quand ?

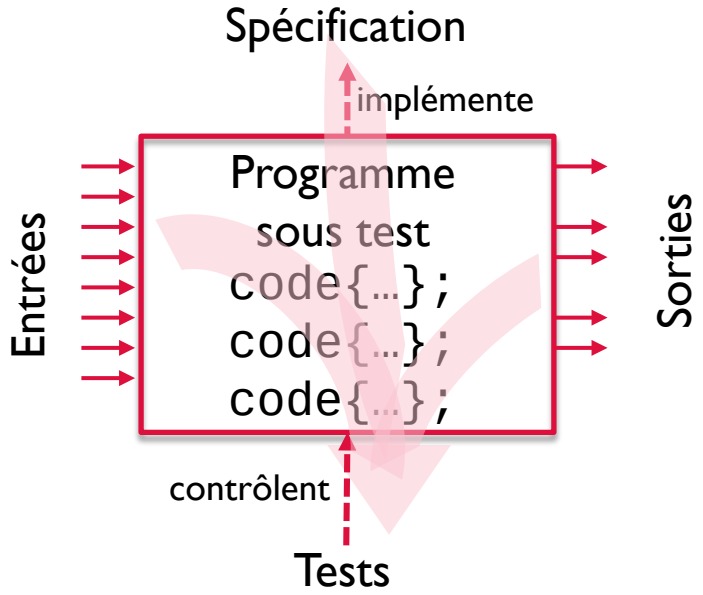
Objectif :
obtenir un ensemble **fini** de cas de test

- Potentiellement il y en a une infinité
  - a = b + c sur $\mathbb{N}$
    - Combien de combinaisons :
      - $2^{32}*2^{32} = 2^{64}$ = un nombre en base décimale avec plus de 18 chiffres !
- Il faut choisir des données
  - Pour atteindre les objectifs :
    - Suffisamment
    - Suffisamment réparties
    - Suffisamment efficaces
  - Tout en respectant les moyens et les délais

La génération de test (rappel CM1)

- Test fonctionnel (test boîte noire)
  - Exploite la description des fonctionnalités du programme
- Test structurel (test boîte blanche)
  - Exploite la structure interne du programme





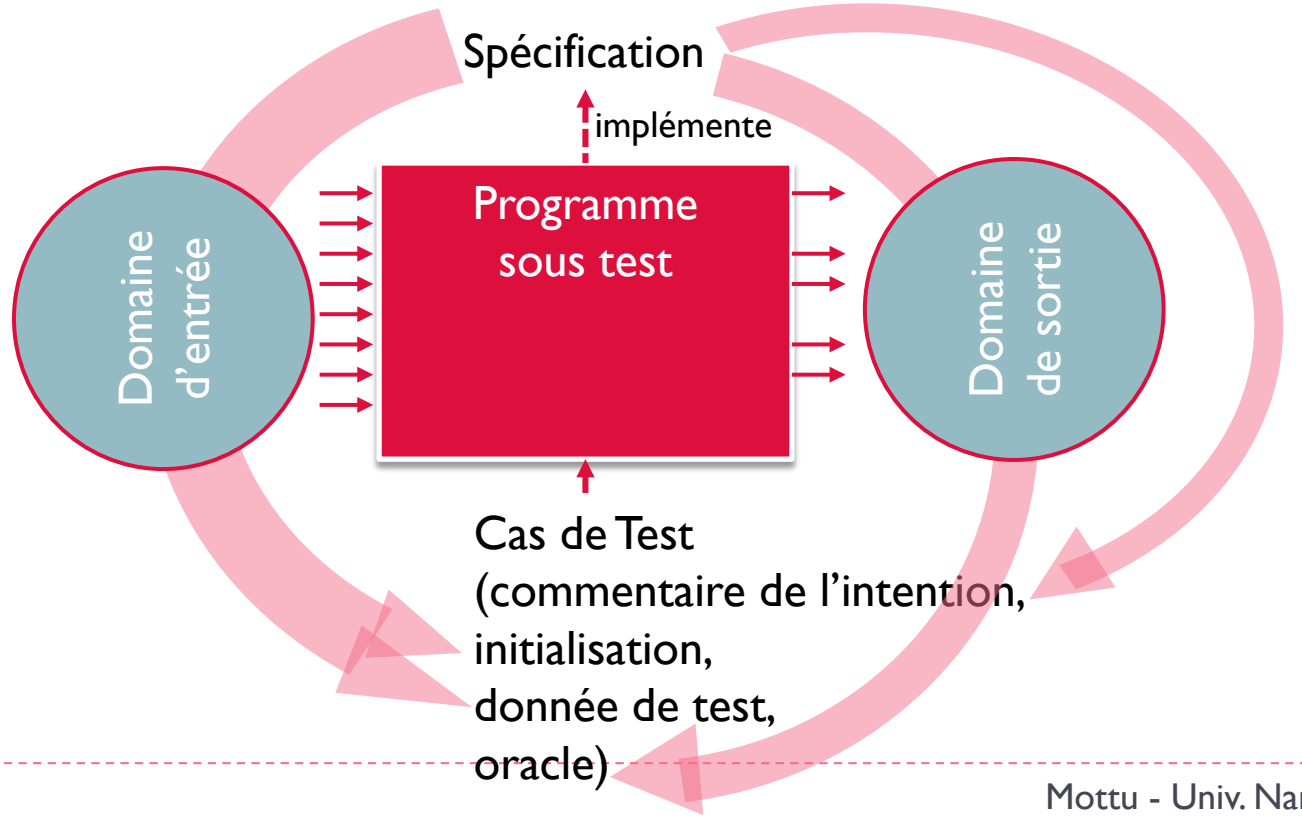
Test fonctionnel en Boite noire

- Le code n'est pas exploité
  - Peut ne même pas être connu

- Basé sur la spécification
  - Exigences (requirements)
  - Modèles (formels ou non)
  - Domaine d'entrée et de sortie
    - interfaces

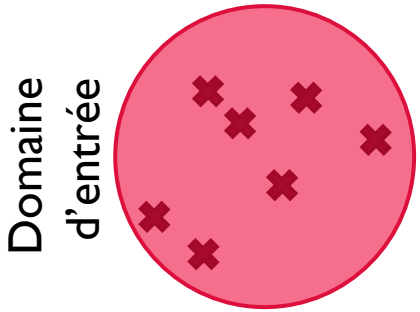
Objectif : sélectionner des données de test dans l'espace de données

- Guidé la conception des cas de test par une sélection des **données de test** :
  - En nombre fini choisies pour leur qualité, leur représentativité, etc.
  - Associer à chaque donnée de test un oracle pour former un cas de test

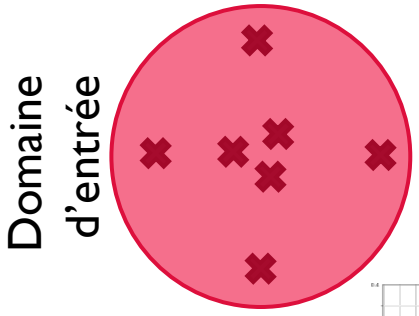


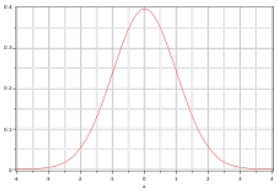
Objectif : sélectionner des données de test dans l'espace de données

1. Génération aléatoire

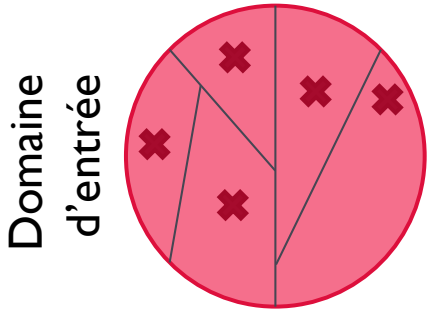


2. Génération statistique

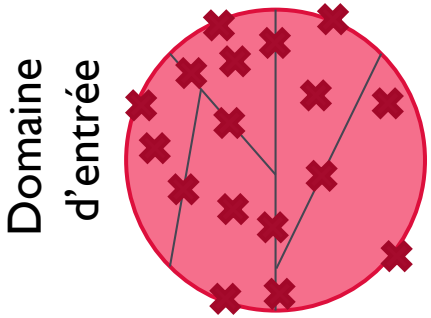




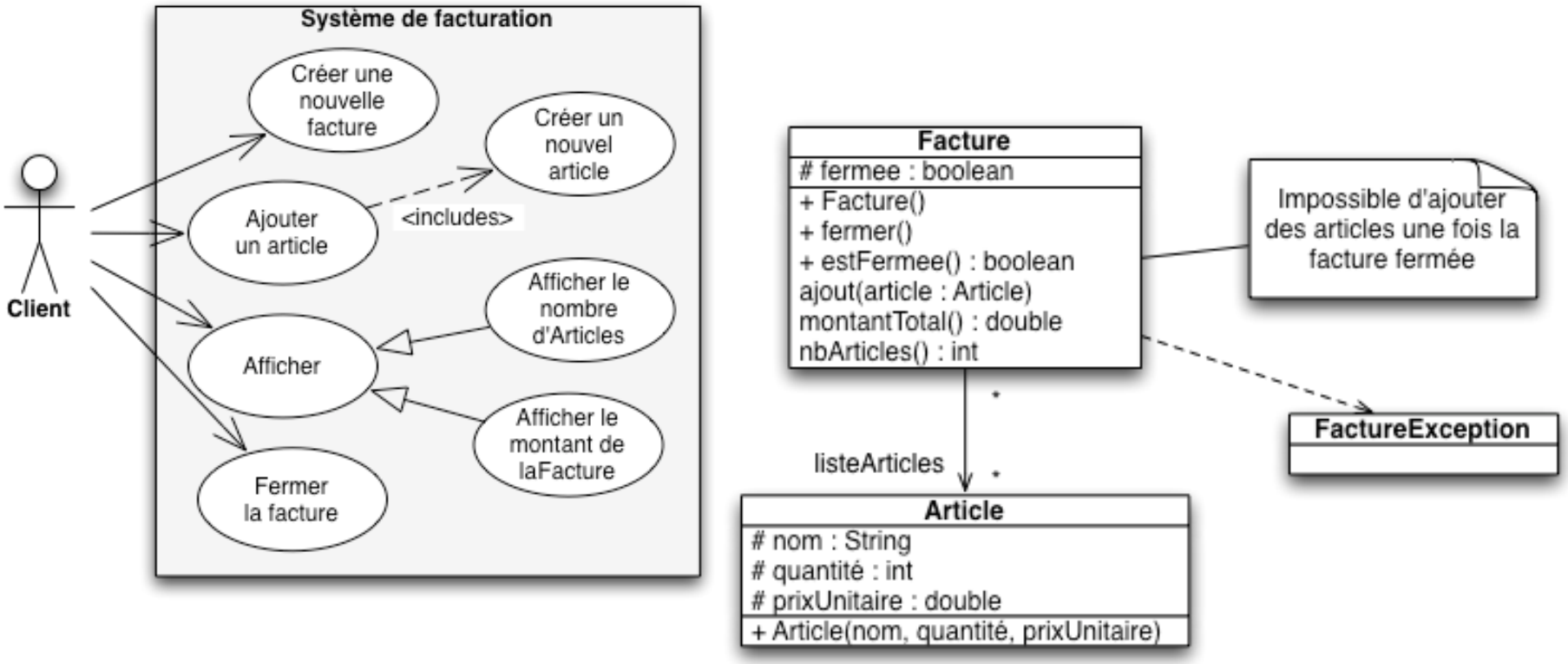
3. Analyse partitionnelle



4. Test aux limites

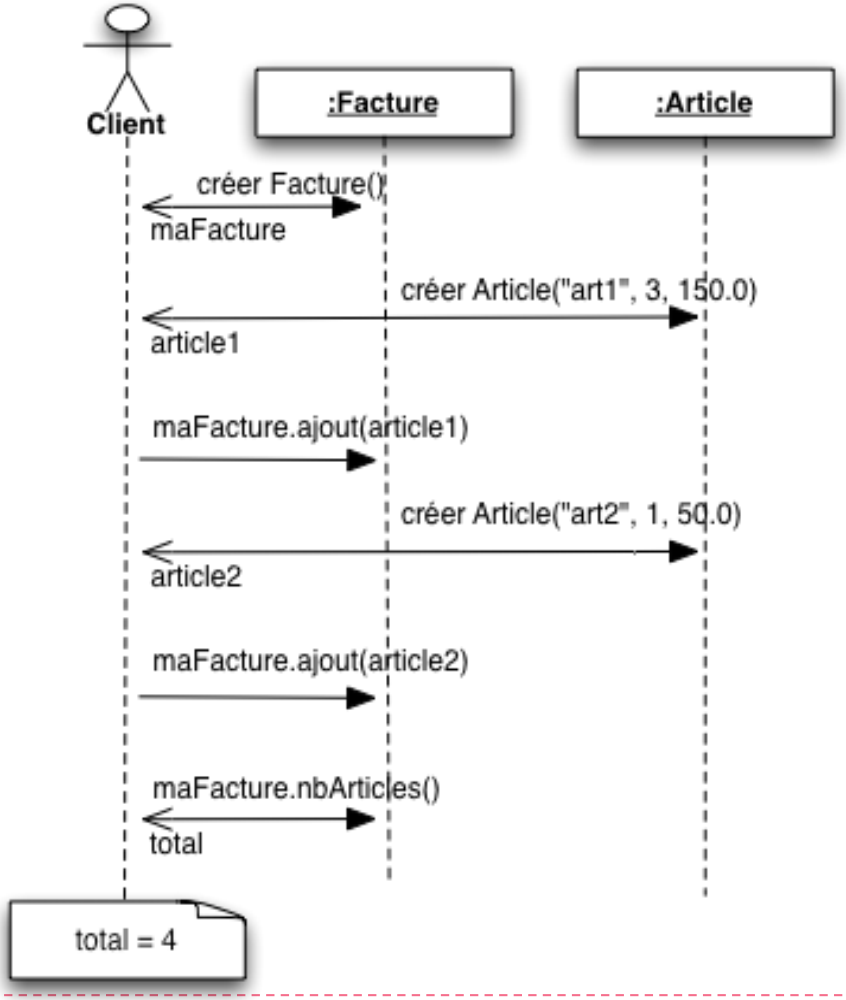


Spécification d'un Gestionnaire « simplifié » de factures

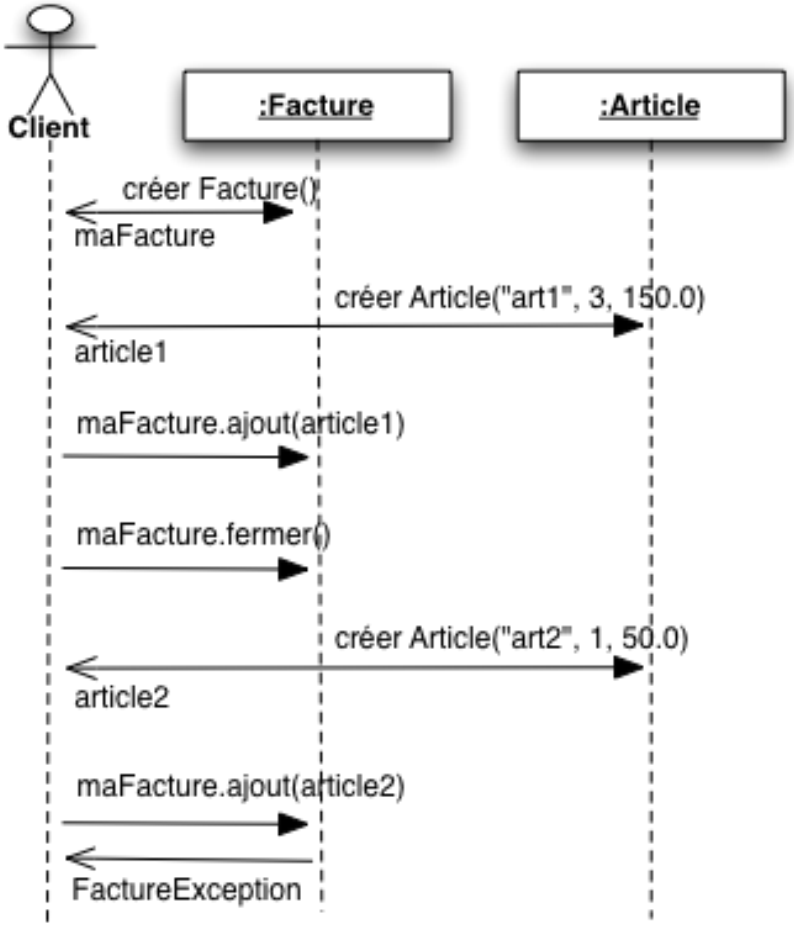


Spécification
Scénarios d'usage :

Nominal :



Exceptionnel :



Domaine d'entrée

- Plusieurs niveaux
  - type des paramètres d'une méthode
  - plage des données
  - pré-condition sur une méthode
    - En particulier sur ses paramètres
  - invariant sur un objet
    - En particulier sur les attributs de la classe
  - ensemble de commandes sur un système
  - grammaire d'un langage
  - ...

- debiter(int : montant)
  - int sur 32bits
  - montant > 0
  - client solvable

  - balance stock/compte créditeur (i.e. pas d'expédition sans argent)

Technique 1: Analyse partitionnelle

- Objectif : créer des classes d'équivalence
  - Une classe d'équivalence est une partie du domaine d'entrée
  - Hypothèse
    - Le comportement de toutes les données d'une même classe d'équivalence est équivalent vis-à-vis des fonctionnalités de la méthode sous test
    - En choisissant une donnée de test par classe d'équivalence, s'il y a une erreur dans ses fonctionnalités alors on la trouve
- Partitionner le domaine d'entrée en classes d'équivalences
  - Partitionner chaque variable formant la donnée de test
  - Chaque classe d'équivalence combine les partitions de chaque variable
  - Choisir une donnée de test dans chaque combinaison de partition

Méthodologie

- Si la valeur à tester appartient à un intervalle : « entier de 1 à 5 »
  - une classe pour les valeurs inférieures « inférieur à 1 »
  - une classe pour les valeurs supérieures « supérieur à 5 »
  - n classes valides « entre 1 et 5 compris »

- Si la donnée est un ensemble de valeurs : « équipe de 3 étudiants »
  - une classe avec pas assez de valeurs : « équipe de moins de 3 étudiants »
  - une classe avec trop de valeurs: « équipe de plus de 3 étudiants »
  - n classes valides : « équipes de 3 étudiants »

- Si la donnée est une contrainte/condition: « rendu avant 17h50 »
  - une classe avec la contrainte respectée « vrai »
  - une classe avec la contrainte non-respectée « faux »

Exemple du nombre de jours

- Soit à tester la méthode :

public static int nbJoursDansMois(int mois, int année)

(la spécification précise que la méthode ne couvre que le XXIème siècle.)

Exemple du nombre de jours

- Soit à tester la méthode :

public static int nbJoursDansMois(int mois, int année)

(la spécification précise que la méthode ne couvre que le XXIème siècle.)

- mois
  - 3 partitions :
    - [-2^31, 1[
    - [1, 12]
    -]12, 2^31 - 1]
- année
  - 3 partitions
    - [-2^31, 2001[
    - [2001, 2100]
    -]2100 ,2^31 - 1]
- 9 classes d'équivalence

- DTs = (-2, 1880), (-2, 2010), (-2, 3000),
  (6, 1880), (6, 2010), (6, 3000),
  (15, 1880), (15, 2010), (15, 3000)

Méthode de l'analyse partitionnelle

- Pour chaque variable formant chaque donnée de test
  1. identifier le type de la variable (int sur 32bits par exemple)
  2. identifier la plage de la variable
     A. Des intervalles nominaux
        - La plage de valeurs du fonctionnement normal (les mois de 1 à 12)
     B. Un/des intervalles exceptionnels
        - La/les plages de valeurs du fonctionnement exceptionnel (<1 et >12)
        - Puisque le type permet de passer ces valeurs, il faut les tester
  3.
  4. choisir une valeur dans chaque intervalle et combiner pour former les données de test

Méthode de l'analyse partitionnelle

- Les données de test des classes d'équivalence ont-elles bien le même comportement attendu ?

- On crée des cas de test
  - Attention à ne pas considérer que les données de test
  - Il faut anticiper l'oracle : vérifiant que le résultat est correct en fonction de la spécification

  - Y a-t-il équivalence fonctionnelle entre
    - nbJoursDansMois(3, 2010)
    - nbJoursDansMois(4, 2010)

Amélioration fonctionnelle

- Mois
  - [-2^31, 1[
  - {1, 3, 5, 7, 8, 10, 12}
  - {4, 6, 9, 11}
  - 2
  -]12 , 2^31 -1]
- Année
  - [-2^31 , 2001[
  - AnneesBissextiles =
    {x ∈ [2001, 2100] :
    (x mod 4 =0 et x mod 100 !=0)
    ou (x mod 400 = 0)}
  - AutresAnnees = [2001, 2100] \ AnneesBissextiles
  -]2100, 2^31 -1]

Méthode de l'analyse partitionnelle

- Pour chaque variable formant chaque donnée de test
  1. identifier le type de la variable (int sur 32bits par exemple)
  2. identifier la plage de la variable
     A. Des intervalles nominaux
        - La plage de valeurs du fonctionnement normal (les mois de 1 à 12)
     B. Un/des intervalles exceptionnels
        - La/les plages de valeurs du fonctionnement exceptionnel (<1 et >12)
        - Puisque le type permet de passer ces valeurs, il faut les tester
  3. **identifier des partitions fonctionnelles**
     A. **Anticipe généralement les combinaisons des variables**
     B. **Etape difficile nécessitant une maîtrise de la spécification**
  4. choisir une valeur dans chaque intervalle et combiner pour former les données de test

Compléter les données de test avec les oracles pour former les cas de test

- Étude du domaine de sortie (oracle)
- Mise en relation des partitions du domaine d'entrée avec les valeurs **attendues** en sortie
  - Attention on ne partitionne pas le domaine de sortie

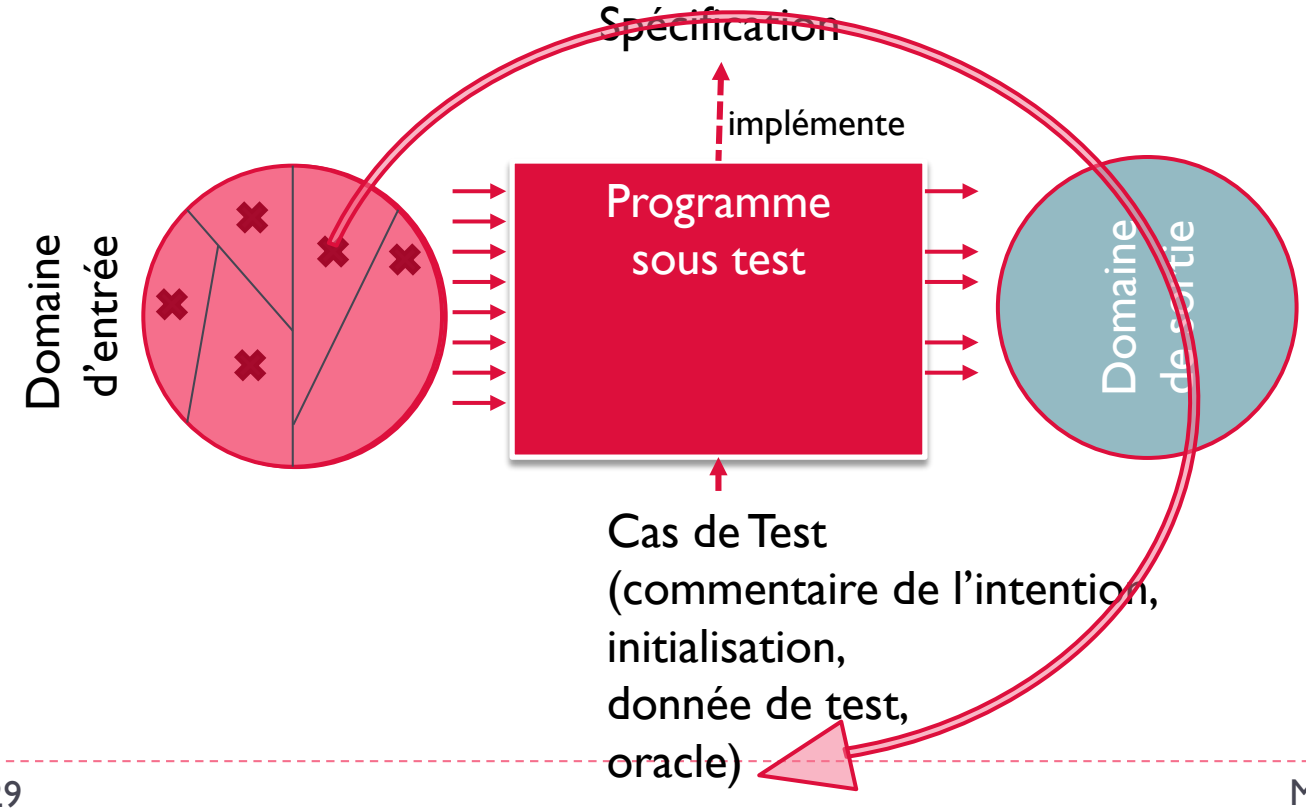


Table de décision

- Finalement, création des cas de test
  - Étude du domaine de sortie (oracle)
  - Mise en relation des partitions du domaine d'entrée avec les valeurs attendues en sortie
    - Attention on ne partitionne pas le domaine de sortie

Table de décision pour gérer plus de 2 dimensions
Exemple du nombre de jour

Entrees	mois	[-2^31, 1[	X									
		{1, 3, 5, 7, 8, 10, 12}					X	X				
		{4, 6, 9, 11}							X	X		
		2									X	X
		]12 , 2^31-1]		X								
	annee	[-2^31,2001[			X							
		AnneesBissextiles					X		X		X	
		AutresAnnees						X		X		X
		]2100 , 2^31-1]				X						
Sortie attendue		31					X	X				
		30							X	X		
		29									X	
		28										X
		entrees invalides	X	X	X	X						

Caractéristiques et Limitation

- le choix des partitions est critique
- possible non prise en compte d'éventuelles différences fonctionnelles entre les éléments appartenant à la même partition
  - l'identification des problèmes/erreurs dépend de ce choix
- partitions hors limites (invalides, exceptionnelles) : tests de robustesse
- partitions dans limites : tests nominaux
- explosion combinatoire des cas de test
  - soit n données d'entrées, et 5 classes : 5^n cas de tests

Technique 2 : Test aux limites

- Intuition:
  - de nombreuses erreurs se produisent dans les cas limites :

  - if (mois < 1)
  - if (mois >=13)
  - for (int i = 0 ; i < length ; i++)
  - etc.

Technique 2 : Test aux limites

- Etape supplémentaire de l'analyse partitionnelle :
  - Pour chaque partition de chaque variable formant une donnée de test
  - déterminer les bornes incluses du domaine
  - Rajouter des partitions sur les bornes et autour
- Exemple
  - pour un intervalle [1, 100] dont les bornes sont incluses car elles sont des valeurs fonctionnellement importantes
  - Aux limites : [0] , [1] , [2] ,]2,99[, [99] , [100], [101]

Sélection des valeurs supplémentaires aux limites

- si x appartient à un intervalle [a; b], prendre
  - les deux valeurs aux limites (a, b)
  - les quatre valeurs a+/-μ, b+/-μ, où μ est le plus petit écart possible
  - une/des valeur(s) dans l'intervalle
- si x appartient à un ensemble ordonne de valeurs, prendre
  - les première, deuxième, avant-dernière, et dernière valeurs
- si x définit un nombre de valeurs, prendre
  - Prendre le minimum de valeurs, le maximum, le minimum-1, le max+1