Test fonctionnel

Jean-Marie Mottu jean-marie.mottu@univ-nantes.fr
Gerson Sunyé gerson.sunye@univ-nantes.fr

TODO

Plan

- Introduction
- Analyse partitionnelle
- Test aux limites
- Pairwise testing
- Autres techniques
- Conclusion

Introduction

Test fonctionnel

« Test basé sur l'analyse de la spécification des fonctionnalités d'un composant ou d'un système »

Cas de test

- Tester c'est exécuter un ensemble de cas de test
- Cas de test
  - Définition du cas de test
  - Initialisation
  - Donnée de test
  - Oracle

  DT x Oracle |-> verdict

  - Verdict : passe, échoue

Cas de test pour l'addition

- Test de l'addition x + y = z
- Cas de test 1
  - Définition du cas de test : vérifier que l'addition de 2 opposées donne 0
  - Initialisation : allumer le programme, etc.
  - Donnée de test : { x = 5; y = -5}
  - Oracle : {0}
- Cas de test 2
  - Définition 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 : {4}

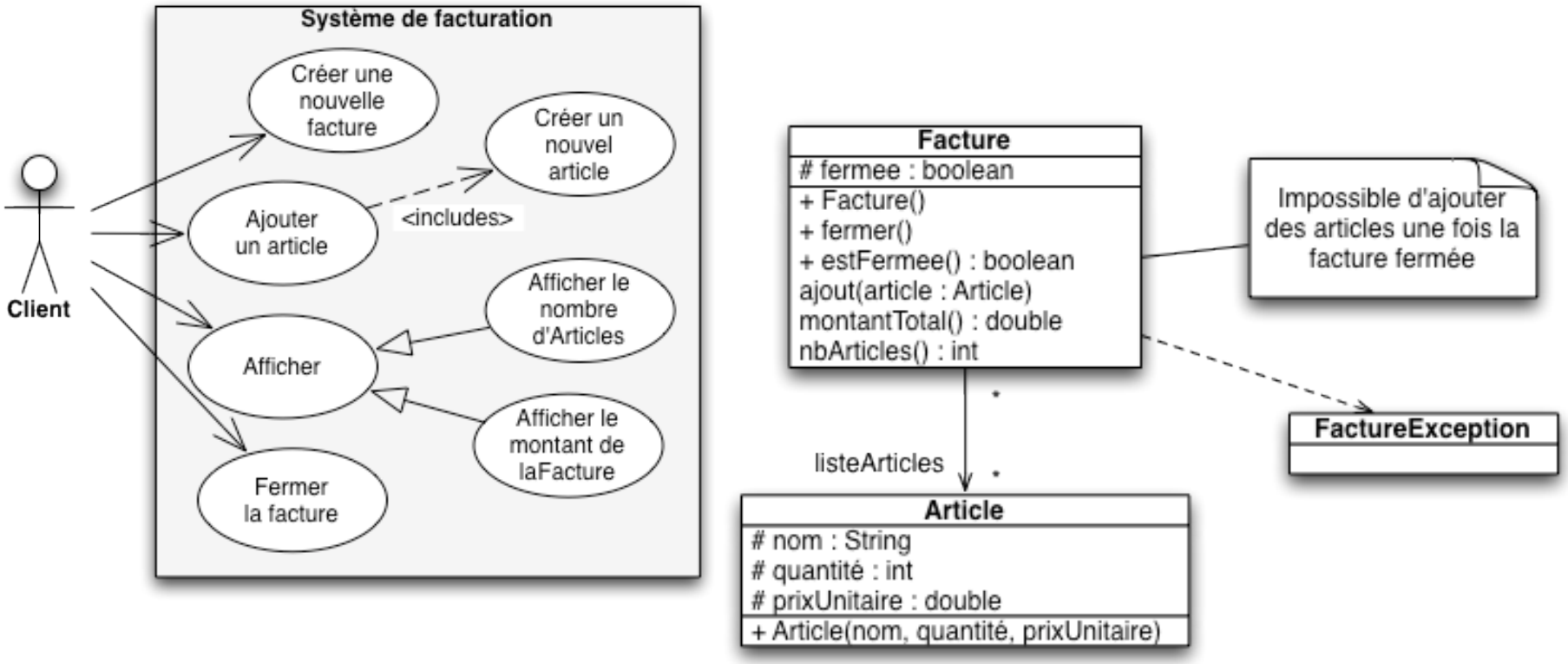
Objectif : obtenir un ensemble de données de test

- Potentiellement il y en a une infinité
  - a = b + c sur $\mathbb{N}$
    - Combien de combinaisons :
      - 2^32*2^32 = 2^64 = un nombre avec plus de 18 chiffres !
- Il faut choisir des données
  - Au mieux :
    - Suffisamment
    - Suffisamment répartie
    - Suffisamment efficace

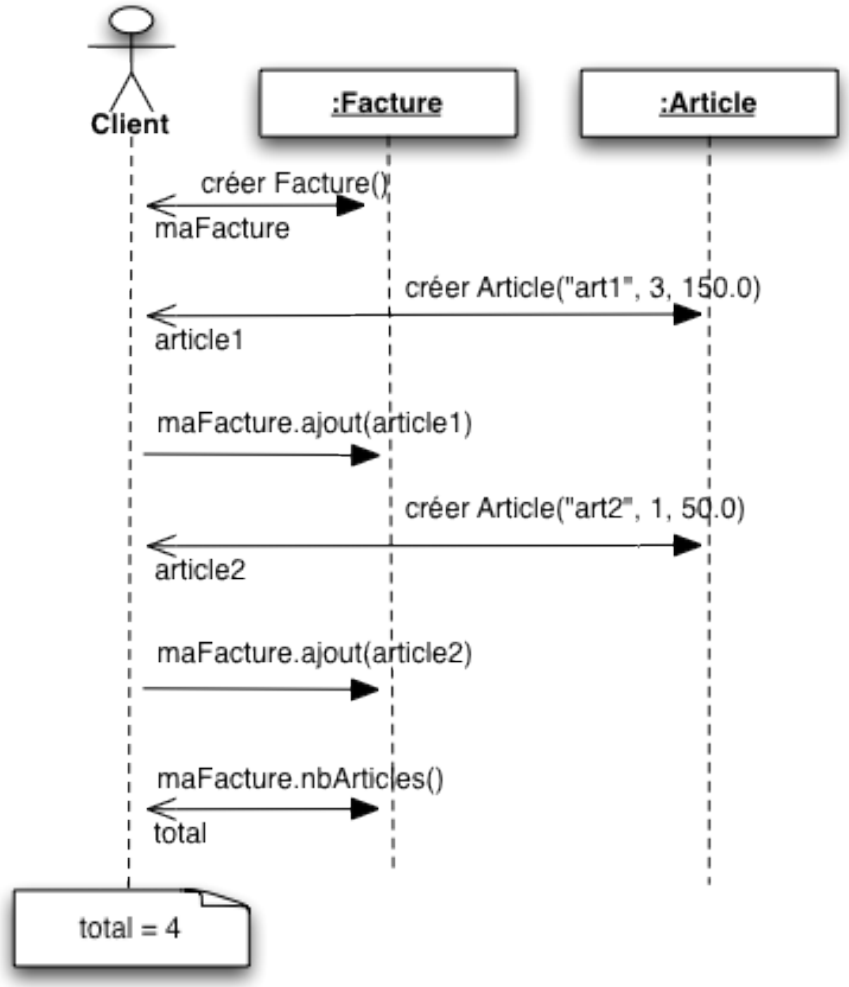
Boite noire

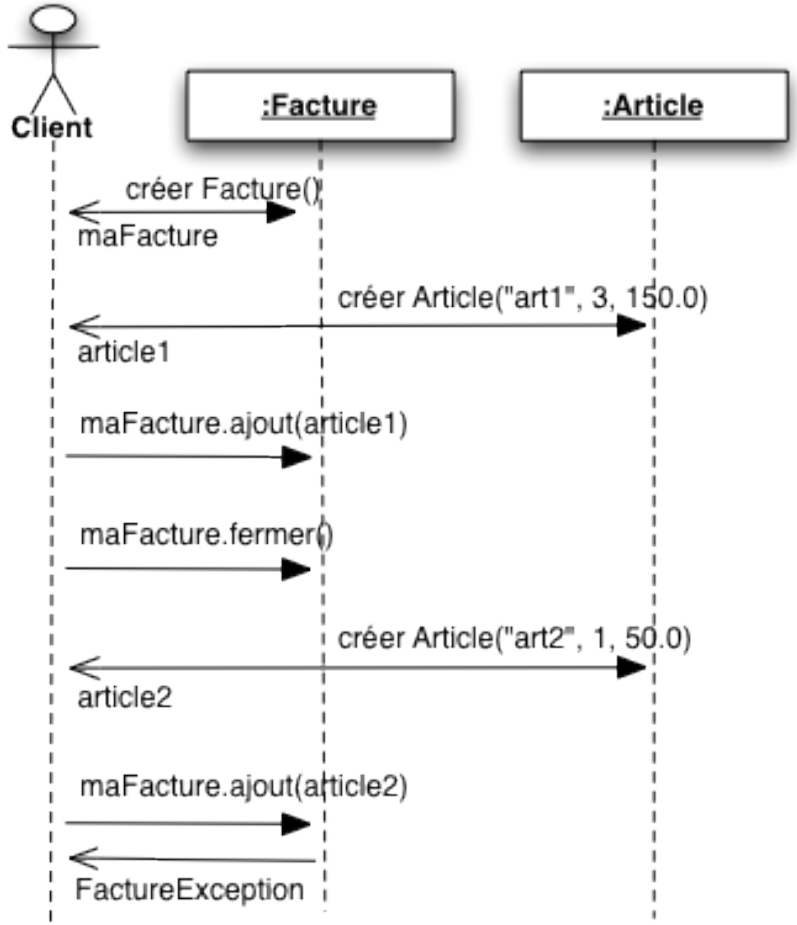
- Le code n'est pas connu
- Basé sur la spécification
  - Règles, modèles (formels ou non)
  - Domaine d'entrée et de sortie.

Spécification d'un Gestionnaire « simplifié » de factures



Spécification
Scénarios d'usage :





Domaine d'entrée

- Plusieurs niveaux
  - type des paramètres d'une méthode
  - pré-condition sur une méthode
  - ensemble de commandes sur un système
  - grammaire d'un langage
  - ...
- On ne peut pas tout explorer, il faut délimiter
  - Génération aléatoire
  - Analyse partitionnelle
  - Test aux limites
  - Graphe causes - effets

Technique 1: Analyse partitionnelle

Analyse partitionnelle

- A partir de la spécification
  - déterminer le domaine d'entrée du programme
- Partitionner le domaine d'entrée en classes d'équivalences
  - identifier des classes d'équivalence pour chaque donnée
  - les classes d'équivalence forment une partition du domaine de chaque donnée en entrée
  - choisir une donnée dans chacune

Méthodologie

- Si la valeur à tester appartient à un intervalle :
  - une classe pour les valeurs inférieures
  - une classe pour les valeurs supérieures
  - n classes valides
- Si la donnée est un ensemble de valeurs :
  - une classe avec l'ensemble vide
  - une classe avec trop de valeurs
  - n classes valides
- Si la donnée est une contrainte/condition:
  - une classe avec la contrainte respectée
  - une classe avec la contrainte non-respectée

Exemple du nombre de jours

- Soit à tester la méthode :

```
public static int nbJoursDansMois(int mois, int
  annee)
```

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

- Mois
  - [-2^31; 0]
  - [1; 12]
  - [13; 2^31 - 1]

Année

[-2^31; 2000]

[2001; 2100]

[2101; 2^31 - 1]

- DTs = (-2, 2010), (3, 2010), (15, 2010), (6, 1880), (6, 3000), (-2, 1880),
- (-2, 3000), (15, 1880), (15, 3000)

Amélioration fonctionnelle

- Mois
  - [-2^31; 0]
  - {1; 3; 5; 7; 8; 10; 12}
  - {4; 6; 9; 11}
  - 2
  - [13; 2^31 -1]

- Année
  - [-2^31; 2000]

  AnneesBissextiles =

  {x ∈ [2001; 2100] : (x mod 4 =0 et x mod 100 !=0) ou (x mod 400 = 0)}
  - AutresAnnees = [2001; 2100] \ AnneesBissextiles
  - [2101; 2^31 -1]

Table de décision

- Élargir l'analyse partitionnelle pour former des cas de test
  - Étude du domaine de sortie (oracle)
  - Mise en relation des partitions du domaine d'entrée et celles du domaine des résultats.

Mois / Année		$[-2^{31}, 2000]$ invalide	$[2001, 2100]$ valide	$[2101, 2^{31}-1]$ invalide
$[-2^{31}, 0]$	invalide	(-4,-10)	(-4,2010)	(-4,2222)
$[1, 12]$	valide	(9,-10)	(9, 2010)	(9,2222)
$[13, 2^{31}-1]$	invalide	(15,-10)	(15,2010)	(15,2222)

Exemple du nombre de jours

- Table de décision

Entrees	mois	$[-2^{31}, 0]$	X									
		$\{1, 3, 5, 7, 8, 10, 12\}$					X	X				
		$\{4, 6, 9, 11\}$							X	X		
		2									X	X
		$[13, 2^{31} - 1]$		X								
	annee	$[-2^{31}, 2001]$			X							
		AnneesBissextiles					X		X		X	
		AutresAnnees						X		X		X
		$[2100, 2^{31} - 1]$				X						
Sortie	31						X	X				
	30								X	X		
	29										X	
	28											X
	entrees invalides		X	X	X	X						

Caractéristiques et limitations

- 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 (invalide) : 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

Test aux limites

- Intuition:
  - de nombreuses erreurs se produisent dans les cas limites
- Pour chaque donnée en entrée
  - déterminer les bornes du domaine
  - prendre des valeurs sur les bornes et juste un peu autour
- Exemple
  - pour un intervalle [1, 100]
  - 1, 100, 2, 99, 0, 101

Sélection des valeurs

- 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- l , le max+ l

Type de donnée

- Booléen : True/False
- Logique Floue: Complètement, Pas du tout, pas complètement, un peu.

Dépendance entre paramètres et partitionnement

- Plusieurs paramètres d'entrée peuvent être partitionnés séparément
- Ces paramètres sont peut-être contraints entres-eux.

- Exemple des triangles:
  - Équilatéral sur [0;5] : (5,5,5), (1,1,1), (1,5,1), etc.

- Contrainte explicite ou implicite
  - Fonctionnel

Pairwise testing

Pairwise testing

- Problématique : explosion combinatoire
  - Méthode sous test avec plusieurs paramètres
    - 5 entiers => chaque entier 3 valeurs => 3^5 valeurs = 243
    - 5 entiers => au limite chaque entier 7 valeurs => 7^5 valeurs = 16807
    - À cela on ajoute l'état du système
  - Au-delà des plages de valeurs des données, la combinatoire complexifie le test

Pairwise testing

- Principe : tester un minimum de fois chaque pair de valeur
- Résultat : réduction du nombre de combinaison

- Combinatoire non définie, approximation :
  - O(nm) quand n et m sont les nombres de possibilités des deux valeurs en ayant le plus

Pairwise testing - exemple

Marque	Carburant	Gamme	Porte
Renault	Essence	Citadine	3
Peugeot	Diesel	Berline	5
Citroen	GPL	Monospace	

- Toutes les combinaisons : 3*3*3*2 = 54
- Toutes les paires : 9

Pairwise testing - exemple

- 9 données de test

DT	Marque	Carburant	Gamme	Porte
1	Renault	Essence	Citadine	3
2	Renault	Diesel	Berline	5
3	Renault	GPL	Monospace	3
4	Peugeot	Essence	Monospace	5
5	Peugeot	Diesel	Citadine	3
6	Peugeot	GPL	Berline	5
7	Citroen	Essence	Berline	3
8	Citroen	Diesel	Monospace	5
9	Citroen	GPL	Citadine	3

  - Toutes les paires

Pairwise testing

- Intuition : la majorité des fautes seront détectées par des combinaisons de 2 variables.
  - Ce n'est pas un postulat
  - Il restera des bugs nécessitant une combinaison exacte de toutes les variables
- Réduction importante, surtout avec beaucoup de variable
- Déclinable avec des triplets, …, jusqu'à la combinatoire
- Largement outillé :
  - http://www.pairwise.org/tools.asp

Autres techniques

Test aléatoire

- Fonction aléatoire
- Efficace pour les fautes provoquant systématiquement des défaillances

- Adapté pour le test de robustesse
  - Combiné avec du partitionnement et de la combinatoire complète

Test statistique

- Utilisation d'une loi statistique
  - Fonction de Gauss …
  - Echantillonnage
- Plusieurs objectifs
  - Trouver là où sont le plus souvent les erreurs
  - Vérifier surtout les valeurs qui seront réellement utilisées.

Conclusion

Conclusion

- Efficacité, facilité de mise en œuvre
- Attention au gain surestimé :
  - Plus on *génère* de données de test, plus on *écrira* des oracles

Références

- « Introduction to Software Testing ». Paul Ammann and Jeff Offutt http://www.cs.gmu.edu/~offutt/softwaretest/
- « Testing Object-Oriented Systems: Models, Patterns, and Tools ». Robert V. Binder.