

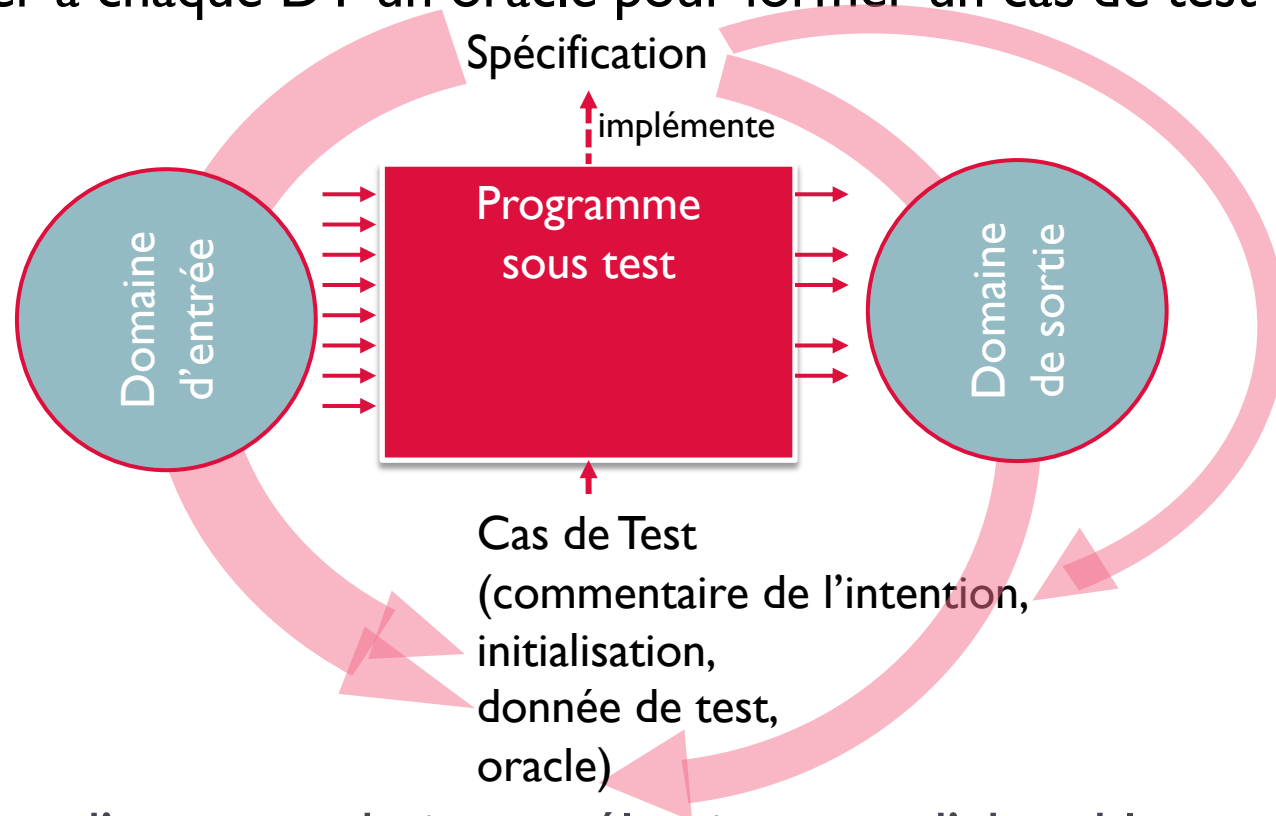
CM6 Conception de tests par approche fonctionnelle

*R2.03 - Qualité de développement 1
Automatisation des tests*

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

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

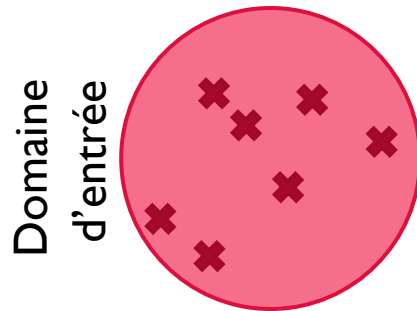
- ▶ Guidé la conception des cas de test par une sélection des **données de test** :
 - ▶ En nombre fini choisies pour leur qualité, leur représentativité, etc.
 - ▶ Associer à chaque DT un oracle pour former un cas de test



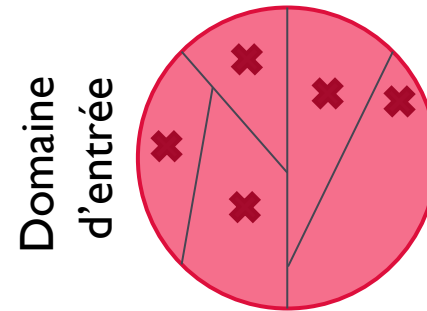
- ▶ Remarque : d'autres techniques sélectionnent d'abord les oracles

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

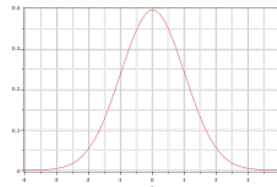
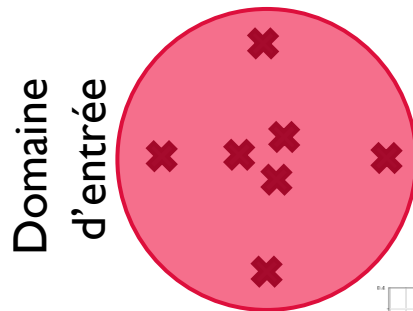
1. Génération aléatoire



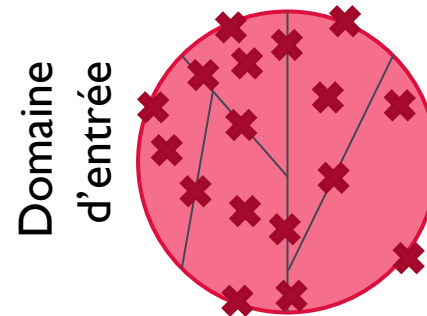
3. Analyse partitionnelle



2. Génération statistique



4. Test aux limites



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 créiteur (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

- ▶ $[-2^{31}, 2001[$
- ▶ $\text{AnneesBissextiles} = \{x \in [2001, 2100] : (x \bmod 4 = 0 \text{ et } x \bmod 100 \neq 0) \text{ ou } (x \bmod 400 = 0)\}$
- ▶ $\text{AutresAnnees} = [2001, 2100] \setminus \text{AnneesBissextiles}$
- ▶ $]2100, 2^{31} - 1]$

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

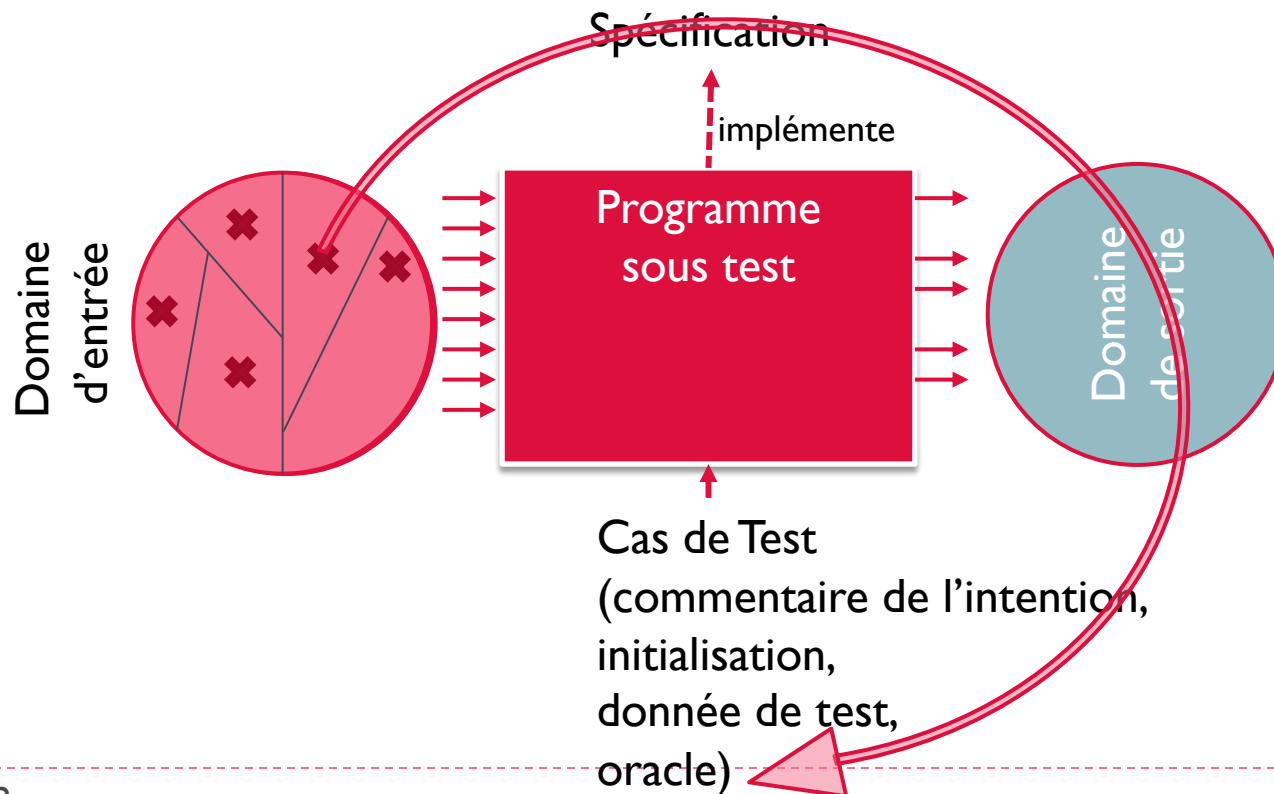


Table de décision

- **Finalelement, 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 <u>attendue</u>	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

Test fonctionnel, pour aller plus loin :

Plus précis, mais plus nombreux : tests aux limites
Moins nombreux, sans perdre trop de précision : n-wise testing

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 $a \pm \mu, b \pm \mu$, où μ est le plus petit écart possible
 - ▶ une/des valeur(s) dans l'intervalle
- ▶ si x appartient à un ensemble ordonné 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

Technique 3 : Pairwise testing

- ▶ **Problématique : explosion combinatoire**
 - ▶ Méthode sous test avec plusieurs variables par donnée de test
 - ▶ 5 entiers \Rightarrow chaque entier 3 valeurs $\Rightarrow 3^5$ valeurs = 243 données
 - ▶ 5 entiers \Rightarrow au limite chaque entier 7 valeurs $\Rightarrow 7^5$ valeurs = 16807 données
 - ▶ Au-delà des plages de valeurs des variables, la combinatoire complexifie le test

Pairwise testing

- ▶ Principe : tester un minimum de fois chaque paire 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 variables en ayant le plus

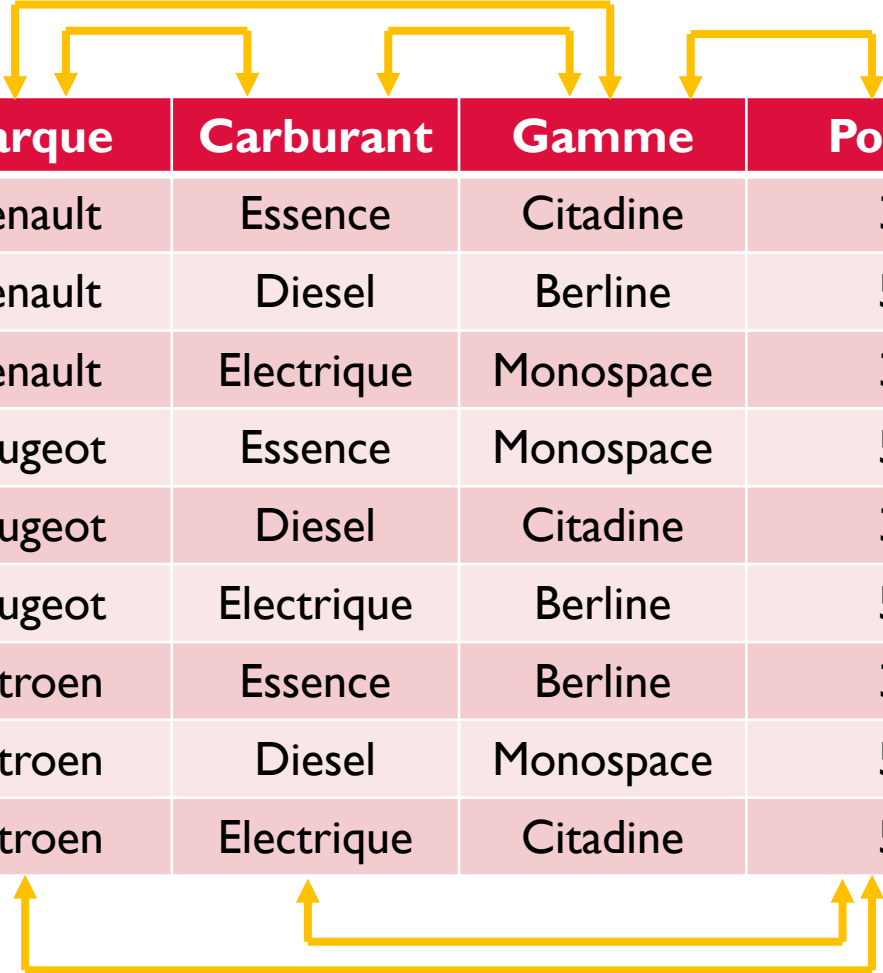
Pairwise testing - exemple

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

- ▶ Toutes les combinaisons : $3*3*3*2 = 54$ combinaisons
- ▶ Toutes les paires : 9 combinaisons au maximum

Pairwise testing - exemple

► 9 données de test



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

- Toutes les paires
- Mais pas toutes les combinaisons
 - eg. pas de Renault, Essence, Berline

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 variables
- ▶ Déclinable avec des triplets, ... , jusqu'à la combinatoire
- ▶ Largement outillé :
 - ▶ <https://www.pairwise.org/tools.html>