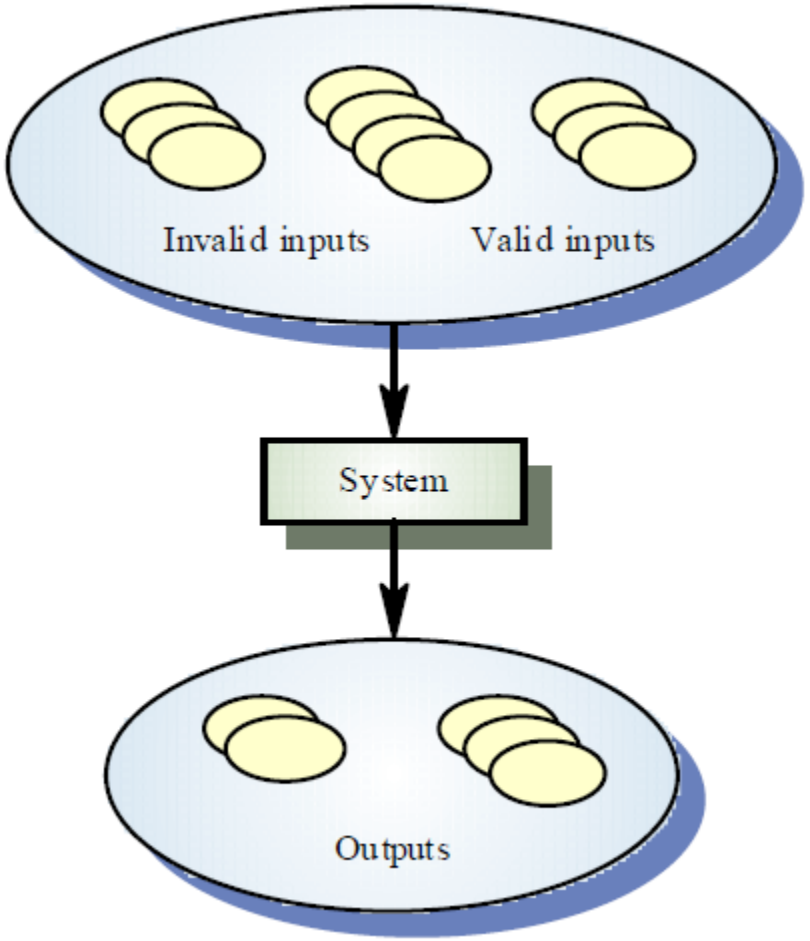
Les tests fonctionnels

Méthodes de test fonctionnel

- Le test fonctionnel vise à examiner le comportement fonctionnel du logiciel et sa conformité avec la spécification du logiciel
  - Sélection des Données de Tests (DT)

- Méthodes du test fonctionnel
  - Analyse partitionnelle des domaines des données d'entrée et test aux limites : test déterministe
  - Test combinatoire –Algorithmes Pairwise
  - Test aléatoire
  - Génération automatique de tests à partir d'une spécification

Analyse partitionnelle des domaines des données d'entrée et test aux limites



Une *classe d'équivalence* correspond à un ensemble de données de tests supposées tester le même comportement, c'est-à-dire activer le même défaut.

Partitionnement de domaines : exemple

- Soit le programme qui permet de calcul de la remise sur un article
  - Pour un prix de vente inférieur à 15,000 DA, pas de remise.
  - Pour un prix de vente inférieur à 20,000 DA, 5% de remise est accordé
  - Pour un prix de vente inférieur à 25,000 DA, 8.5% de remise est accordé
- Quatre classes d'équivalances valides avec des données en entrée correctes peuvent être définies

Parameter	Equivalence classes	Representative
Sales price	vEC1: $0 \leq x < 15000$	14500
	vEC2: $15000 \leq x \leq 20000$	16500
	vEC3: $20000 < x < 25000$	24750
	vEC4: $x \geq 25000$	31800

Partitionnement de domaines : exemple

- En plus des valeurs en entrée correctes, les valeurs ‘entrée incorrectes doivent être testées
- Les classes d’équivalence pour ces valeurs doivent être définies et des tests avec des représentants dé ces classes doivent être exécutés.

Parameter	Equivalence classes	Representative
Sales price	iEC1: x < 0 negative, i.e., wrong sales price	-4000
	iEC2: x > 1000000 unrealistically high sales price[a]	1500800

Règles de partitionnement des domaines

- Si la valeur appartient à un intervalle, construire :
  - une classe pour les valeurs inférieures,
  - une classe pour les valeurs supérieures,
  - n classes valides.
- Si la donnée est un ensemble de valeurs, construire :
  - une classe avec l'ensemble vide,
  - une classe avec trop de valeurs,
  - n classes valides.
- Si la donnée est une obligation ou une contrainte (forme, sens, syntaxe), construire :
  - une classe avec la contrainte respectée,
  - une classe avec la contrainte non-respectée

Analyse partitionnelle-Méthode

- Trois phases :
  - Pour chaque donnée d'entrée, calcul de classes d'équivalence sur les domaines de valeurs,
  - Choix d'un représentant de chaque classe d'équivalence,
  - Composition par produit cartésien sur l'ensemble des données d'entrée pour établir les DT.

- Soit Ci, une classe d'équivalence,
- $\cup Ci = E \wedge \forall i,j Ci \cap Cj = \emptyset$

Analyse partitionnelle-Méthode

Règles concernant les tests:

- Les représentants des classes d‘équivalence valides peuvent être combinés: toutes les combinaisons possibles de classes d’équivalence valides doivent être couvertes.
  - Chacune de ces combinaisons construit un cas de test valide ou un cas de test positif.
- Le représentant d’une classe d’équivalence invalide doit être combiné seulement avec les représentants des classes d’équivalence valides.
  - Pour chaque classe d’équivalence invalide, un cas de test négatif sera ajouté.

Analyse partitionnelle-Méthode

Réduction du nombre de cas de tests:

- Le nombre de cas de test valides est le produit du nombre de classes d'équivalence par paramètre.
- Ceci peut générer des centaines de cas de tests valides pour quelques paramètres.
- Aussi, il est nécessaire de définir plus de règles pour réduire ce nombre.
  - Combiner les cas de test et les trier par fréquence d'occurrence (profile d'utilisation) . Seules les cas de tests pertinents sont testés.
  - Les cas de test contenant des valeurs aux limites ou des combinaisons de valeurs aux limites sont favorisés.
  - Combiner chaque représentant d'une classe d'équivalence avec chaque représentant des autres classes d'équivalence (pairwise)
  - S'assurer que chaque représentant d'une classe d'equivalence apparait dans au moins un cas de test. C'est un critère minimum.

Exemple

- Soit le programme qui permet le calcul prix total de vente d'un véhicule avec les paramètres suivants:

```
double calculate_price (
double baseprice,      // base price of the vehicle
double specialprice,   // special model addition
double extraprice,     // price of the extras
int extras,            // number of extras
double discount        // dealer's discount
)
```

Exemple

- Etape 1: identifier le domaine

Parameter	Equivalence classes
baseprice	vEC_{11}: [MIN_DOUBLE, … , MAX_DOUBLE] iEC_{11}: NaN
specialprice	vEC_{21}: [MIN_DOUBLE, … , MAX_DOUBLE] iEC_{21}: NaN
extraprice	vEC_{31}: [MIN_DOUBLE, … , MAX_DOUBLE] iEC_{31}: NaN
extras	vEC_{41}: [MIN_INT, … , MAX_INT] iEC_{41}: NaN
discount	vEC_{51}: [MIN_DOUBLE, … , MAX_DOUBLE] iEC_{51}: NaN

Exemple

- Etape 2: Affiner les classes d'équivalence par rapport aux spécifications

Parameter	Equivalence classes	Representatives
baseprice	vEC_{11}: [0, … , MAX_DOUBLE]	20000.00
	iEC_{11}: [MIN_DOUBLE, … , 0[[a] iEC_{12}: NaN	-1.00 "abc"
specialprice	vEC_{21}: [0, … , MAX_DOUBLE]	3450.00
	iEC_{21}: [MIN_DOUBLE, … , 0[iEC_{22}: NaN	-1.00 "abc"
extraprice	vEC_{31}: [0, … , MAX_DOUBLE]	6000.00
	iEC_{31}: [MIN_DOUBLE, … , 0[iEC_{32}: NaN	-1.00 "abc"
extras	vEC_{41}: [0, … , 2] vEC_{42}: [3, 4] vEC_{43}: [5, … , MAX_INT]	1 3 20
	iEC_{41}: [MIN_INT, … , 0[iEC_{42}: NaN	-1.00 "abc"
discount	vEC_{51}: [0, … , 100]	10.00
	iEC_{51}: [MIN_DOUBLE, … , 0[iEC_{52}:]100, … , MAX_DOUBLE] iEC_{53}: NaN	-1.00 101.00 "abc"

Exemple

- Etape 3: Sélectionner des représentants

Parameter	Equivalence classes	Representatives
baseprice	vEC_{11}: [0, … , MAX_DOUBLE]	20000.00
	iEC_{11}: [MIN_DOUBLE, … , 0[[a] iEC_{12}: NaN	-1.00 "abc"
specialprice	vEC_{21}: [0, … , MAX_DOUBLE]	3450.00
	iEC_{21}: [MIN_DOUBLE, … , 0[iEC_{22}: NaN	-1.00 "abc"
extraprice	vEC_{31}: [0, … , MAX_DOUBLE]	6000.00
	iEC_{31}: [MIN_DOUBLE, … , 0[iEC_{32}: NaN	-1.00 "abc"
extras	vEC_{41}: [0, … , 2] vEC_{42}: [3, 4] vEC_{43}: [5, … , MAX_INT]	1 3 20
	iEC_{41}: [MIN_INT, … , 0[iEC_{42}: NaN	-1.00 "abc"
discount	vEC_{51}: [0, … , 100]	10.00
	iEC_{51}: [MIN_DOUBLE, … , 0[iEC_{52}:]100, … , MAX_DOUBLE] iEC_{53}: NaN	-1.00 101.00 "abc"

Exemple

Etape 4: Combiner les cas de test

- Le nombre de cas de test valides est:

  $1 \times 1 \times 1 \times 3 \times 1 = 3$ *cas d tests valides*

- Le nombre de cas de test invalides est

  $2 + 2 + 2 + 2 + 3 = 11$ *cas de test invalids*

- Au total, 14 cas de tests sont définis à partir des 18 clases d'equivalence définies

Exemple

Etape 4: Combiner les cas de test

	Parameter					
Test case	baseprice	special price	extraprice	extras	discount	result
1	20000.00	3450.00	6000.00	1	10.00	27450.00
2	20000.00	3450.00	6000.00	3	10.00	26850.00
3	20000.00	3450.00	6000.00	20	10.00	26550.00
4	-1.00	3450.00	6000.00	1	10.00	NOT_VALID
5	"abc"	3450.00	6000.00	1	10.00	NOT_VALID
6	20000.00	-1.00	6000.00	1	10.00	NOT_VALID
7	20000.00	"abc"	6000.00	1	10.00	NOT_VALID
8	20000.00	3450.00	-1.00	1	10.00	NOT_VALID
9	20000.00	3450.00	"abc"	1	10.00	NOT_VALID
10	20000.00	3450.00	6000.00	-1.00	10.00	NOT_VALID
11	20000.00	3450.00	6000.00	"abc"	10.00	NOT_VALID
12	20000.00	3450.00	6000.00	1	-1.00	NOT_VALID
13	20000.00	3450.00	6000.00	1	101.00	NOT_VALID
14	20000.00	3450.00	6000.00	1	"abc"	NOT_VALID