Concevoir des tests grâce aux spécifications

Yassamine Seladji

yassamine.seladji@gmail.com

24 septembre 2017

Introduction

Le rôle du testeur est de trouver les jeux de valeurs qui permettrons de découvrir les comportement incorrecte du logiciel.
Les tests fonctionnels ou tests en boite noire.

Réduire le combinatoire

- Énumérer toutes les combinaisons possibles des valeurs des paramètres en entrée.

Réduire le combinatoire par l'exemple

Comment tester efficacement un composant logiciel qui génère des scripts utilisés pour valider le comportement de serveur web ?
Ce composant prend quatre paramètres en entrée et génère un script adapté à ces valeurs de paramètres.
Les paramètres sont :

- Le type du poste client a partir duquel le script sera exécuter, les valeurs possibles sont :
  - Linux, Solaris, Windows, Vista, Mac OS.
- Le type du navigateur client, les valeurs possibles sont :
  - Firefox, Internet Explorer, Netscape.
- Le type de données téléchargées, les valeurs possibles sont :
  - jpeg, avi, mp3, wave.
- Le fait d'utiliser une connexion sécurisée ou non.

Il faut réaliser 5*3*4*2 = 120 tests.
Si un test prend une demi-heure, il faut 60 heures pour tester ce composant.

Réduire le combinatoire : All singles

- Construire des jeux de tests couvrant au moins chaque possibilité pour chaque paramètre.
- Le nombre de tests représente le cardinal du plus grand ensemble de valeurs.

Réduire le combinatoire : All singles par l'exemple

- On commence par le plus grand ensemble : l'ensemble des systèmes possibles, donc de cardinalité 5.
  - (Windows 7, ?, ?, ?)
  - (Windows vista, ?, ?, ?)
  - (Linux, ?, ?, ?)
  - (Solaris, ?, ?, ?)
  - (Mac OS, ?, ?, ?)

Réduire le combinatoire : All singles par l'exemple

- On commence par le plus grand ensemble : l'ensemble des systèmes possibles, donc de cardinalité 5.
- On ajoute les valeurs du second paramètre : l'ensemble des navigateurs web.
  - (Windows 7,Internet Explorer, ?, ?)
  - (Windows vista,Netscape, ?, ?)
  - (Linux, Firefox, ?, ?)
  - (Solaris, Netscape, ?, ?)
  - (Mac OS,Firefox, ?, ?)

Réduire le combinatoire : All singles par l'exemple

- On commence par le plus grand ensemble : l'ensemble des systèmes possibles, donc de cardinalité 5.
- On ajoute les valeurs du second paramètre : l'ensemble des navigateurs web.
- Utiliser le même principe pour les autres paramètres.
  - (Windows 7, Internet Explorer,jpeg, sécurisée)
  - (Windows vista, Netscape,avi, non sécurisée)
  - (Linux, Firefox,mp3, sécurisée)
  - (Solaris, Netscape,mp3, sécurisée)
  - (Mac OS, Firefox,avi, non sécurisée)

Réduire le combinatoire : All singles par l'exemple

- On commence par le plus grand ensemble : l'ensemble des systèmes possibles, donc de cardinalité 5.
- On ajoute les valeurs du second paramètre : l'ensemble des navigateurs web.
- Utiliser le même principe pour les autres paramètres.
  - (Windows 7, Internet Explorer, jpeg, sécurisée)
  - (Windows vista, Netscape, avi, non sécurisée)
  - (Linux, Firefox, mp3, sécurisée)
  - (Solaris, Netscape, mp3, sécurisée)
  - (Mac OS, Firefox, avi, non sécurisée)

Remarque : Si on à le choix entre deux valeurs, on choisis celle qui semble la plus représentative des cas d'utilisation réel du composant.

Réduire le combinatoire : All singles

Les avantages :

- Réduire le nombre de jeux tests.
- Maintenir une certaine qualité des tests : toute valeur d'un paramètre est testée au moins une fois.
- La détection facile d'une grosse erreur ou d'un oublie de réalisation.

Les inconvénients :

- Difficile de détecter les erreurs liées à une association particulière de paramètre.

Réduire le combinatoire : All pairs

- Toutes les paires de possibilités soient couvertes au moins par un jeux de tests.
  Exemple : (système/navigateur), (système/type de fichier), (navigateur/type de fichier).
- Ordonner en ordre décroissant les variables par rapport à leur nombre de valeurs.
  Exemple : systèmes (5 valeurs), données (4 valeurs), navigateurs (3 valeurs) et connexion (2 valeurs).
- Construire une table avec une colonne par variable et au moins V1 $\times$ V2 lignes.
  Exemple : une table avec 4 colonnes et $5 \times 4 = 20$ lignes.

tablePair2.png

Réduire le combinatoire : All pairs

- Mettre dans les V2 premières lignes la première valeur du premier paramètre, puis le second paramètre dans les V2 lignes suivantes et ainsi de suite.
- Alterner dans la seconde colonne les V2 valeurs du second paramètre (ex : type de fichier).

Système	Type de fichiers	Navigateur	Connexion
Windows Me	Jpeg		
Windows Me	Avi		
Windows Me	Wave		
Windows Me	Mp3		
VISTA	Jpeg		
VISTA	Avi		
VISTA	Wave		
VISTA	Mp3		
Linux	Jpeg		
Linux	Avi		
Linux	Wave		
Linux	Mp3		
Mac Os X	Jpeg		
Mac Os X	Avi		
Mac Os X	Wave		
Mac Os X	Mp3		
Solaris	Jpeg		
Solaris	Avi		
Solaris	Wave		
Solaris	Mp3		

Réduire le combinatoire : All pairs

- Alterner dans la reste des colonnes les différentes valeurs des paramètres restants.

Système	Type de fichiers	Navigateur	Connexion
Windows Me	Jpeg	Internet Explorer	sécurisée
Windows Me	Avi	Netscape	non sécurisée
Windows Me	Wave	FireFox	sécurisée
Windows Me	Mp3	Internet Explorer	non sécurisée
VISTA	Jpeg	Netscape	non sécurisée
VISTA	Avi	FireFox	sécurisée
VISTA	Wave	Internet Explorer	non sécurisée
VISTA	Mp3	Netscape	sécurisée
Linux	Jpeg	Netscape	non sécurisée
Linux	Avi	FireFox	sécurisée
Linux	Wave	Internet Explorer	non sécurisée
Linux	Mp3	Netscape	sécurisée
Mac Os X	Jpeg	FireFox	sécurisée
Mac Os X	Avi	Internet Explorer	non sécurisée
Mac Os X	Wave	Netscape	sécurisée
Mac Os X	Mp3	FireFox	non sécurisée
Solaris	Jpeg	Internet Explorer	non sécurisée
Solaris	Avi	Netscape	sécurisée
Solaris	Wave	FireFox	non sécurisée
Solaris	Mp3	Internet Explorer	sécurisée

Réduire le combinatoire : All pairs

Les avantages :

- Technique très efficaces.
- Simple à mettre en place.
- Large couverture de cas de tests.

Les inconvénients :

- Pour des paramètres de cardinalité très élevé, cette technique ne fonctionne pas.

Le test par classes d'équivalence

- Une classe d'équivalence est un ensemble de valeurs pour lesquelles on ne peut distinguer le comportement du logiciel.
- Les spécifications sont utilisées pour définir les classes d'équivalence valides et invalides.

Le test par classes d'équivalence

- Une classe d'équivalence est un ensemble de valeurs pour lesquelles on ne peut distinguer le comportement du logiciel.
- Les spécifications sont utilisées pour définir les classes d'équivalence valides et invalides.

Exemple : une fonction qui prends en paramètre le numéro de la wilaya et retourne son nombre d'habitants.

- La classe d'équivalence valide : numéro de wilaya entre 1 et 48.
- La classe d'équivalence invalide : numéro de wilaya inférieur à 1 ou supérieur à 48.

Le test par classes d'équivalence

- toutes les valeurs d'une classe donnée conduisent au même comportement du logiciel.

Le test par classes d'équivalence

- toutes les valeurs d'une classe donnée conduisent au même comportement du logiciel.

Exemple : une fonction qui prends en paramètre le numéro de la wilaya et retourne son nombre d'habitants.

Validité des entrées	Classes d'équivalence	Données de test
Entrées valides	[1 - 48]	13
Entrées invalides	[minInt – 1 [	-30
Entrées invalides	] 48 – MaxInt]	100

La construction des classes d'équivalence

- Si une condition d'entrée ou de sortie définit un intervalle de valeurs :
  - la classe d'équivalence valide : une valeur dans l'intervalle.
  - deux classes d'équivalence invalides : une à chaque bout de l'intervalle.
- Exemple : Le numéro de wilaya est compris entre 1 et 48.

La construction des classes d'équivalence

- Si une condition d'entrée ou de sortie définit **N** valeurs :
  - la classe d'équivalence valide : elle représente les éléments à exactement **N** valeurs.
  - deux classes d'équivalence invalides : un représentant les éléments à moins de **N** valeurs, et un représentant les éléments à plus de **N** valeurs
- Exemple : un tableau de valeurs.

La construction des classes d'équivalence

- Si une condition d'entrée ou de sortie définit un ensemble de valeurs :
  - la classe d'équivalence valide : elle représente les valeurs dans l'ensemble prédéfini ou bien une classe par valeur si le programme les différencie.
  - une classe d'équivalence invalide : représentant les valeurs hors de l'ensemble.
- Exemple : une énumération de valeurs.

La construction des classes d'équivalence

- Si une condition d'entrée ou de sortie définit une contrainte devant être vérifiée :
  - la classe d'équivalence valide : la condition est vérifiée.
  - une classe d'équivalence invalide : la condition n'est pas vérifiée.
- Exemple : le premier caractère d'un identifiant devant être une lettre.

La construction des classes d'équivalence : par l'exemple

Tester une fonction *Lendemain* qui calcule le lendemain d'une date passée en paramètre et définie par trois entiers : jour, mois, année.
Les spécifications données sont les suivantes :

- l'année doit être supérieur à 1582 et inférieur à 3000.
- prendre en compte les années bissextiles.
- le nombre de jours du mois de février d'une année bissextile est de 29 jours.

La construction des classes d'équivalence : par l'exemple

Tester une fonction *Lendemain* qui calcule le lendemain d'une date passée en paramètre et définie par trois entiers : jour, mois, année.

Utiliser les contraintes élémentaires pour construire les classes d'équivalence :

Classes d'équivalences valides	Classes d'équivalences invalides
Jour $\in$ [1, 31]	Jour < 1
	Jour > 31
Mois $\in$ [1, 12]	Mois < 1
	Mois > 12
Année $\in$ [1582, 3000]	Année < 1582
	Année > 3000

La construction des classes d'équivalence : par l'exemple

Tester une fonction *Lendemain* qui calcule le lendemain d'une date passée en paramètre et définie par trois entiers : jour, mois, année.

Une classe d'équivalence complexe pouvant faire apparaitre des comportements distincts doit être découper en classes plus petites.

Classes d'équivalences valides	
Jour $\in$ [1, 31]	
Mois $\in$ [1, 12]	Mois $\in \{1, 3, 5, 7, 8, 10, 12\}$
	Mois $\in \{4, 6, 9, 11\}$
	Mois = 2
Année $\in$ [1582, 3000]	Année bissextile dans l'intervalle [1582, 3000]
	Année non bissextile dans l'intervalle [1582, 3000]
	Année = 2000

La construction des classes d'équivalence : par l'exemple

Tester une fonction *Lendemain* qui calcule le lendemain d'une date passée en paramètre et définie par trois entiers : jour, mois, année.

Ajouter les contraintes liant les entrées et les sorties (aspect fonctionnel)

- valeur de jour correspondant à une fin du mois, ou valeur du mois correspondant à une fin d'année.

Classes d'équivalences valides		
Jour $\in$ [1, 31]	Jour $\in$ [1, 28]	
	Jour = 29	
	Jour = 30	
	Jour = 31	
Mois $\in$ [1, 12]	Mois $\in \{1, 3, 5, 7, 8, 10, 12\}$	Mois $\in \{1, 3, 5, 7, 8, 10\}$
		Mois = 12
	Mois $\in \{4, 6, 9, 11\}$	
	Mois = 2	
Année $\in$ [1582, 3000]	Année bissextile dans l'intervalle [1582, 3000]	
	Année non bissextile dans l'intervalle [1582, 3000]	
	Année = 2000	

La construction des classes d'équivalence : par l'exemple

Tester une fonction *Lendemain* qui calcule le lendemain d'une date passée en paramètre et définie par trois entiers : jour, mois, année.

Remarque :)

- Certaines combinaisons de valeurs valides peuvent donner des entrées invalides :
  - 29-02-2004
  - 31-04-2007

La construction des classes d'équivalence : par l'exemple

La génération des jeux de tests : classes d'équivalence + all singles

- Les classes d'équivalence valides :

Jeux de valeurs	Résultat attendu	Classes d'équivalences couvertes
(14, 7, 2008)	(15, 7, 2008)	Jour $\in$ [1, 28] Mois $\in$ {1, 3, 5, 7, 8, 10} Année bissextile
(29, 12, 1997)	(30, 12, 1997)	Jour = 29 Mois = 12, Année non bissextile
(30, 4, 2000)	(31, 4, 2000)	Jour = 30 Mois $\in$ {4, 6, 9, 11} Année = 2000
(31, 12, 1999)	(1, 1, 2000)	Jour = 31 Mois = 12 Année non bissextile

La construction des classes d'équivalence : par l'exemple

La génération des jeux de tests : classes d'équivalence + all singles

- Les classes d'équivalence invalides :

Jeux de valeurs	Classes d'équivalences couvertes
(-10, 2, 2000)	Jour < 1
(33, 12, 2001)	Jour > 31
(2, 0, 1999)	Mois < 1
(3, 14, 1997)	Mois > 12
(1, 1, 1500)	Année < 1582
(1, 1, 4000)	Année > 3000

La construction des classes d'équivalence : par l'exemple

La génération des jeux de tests : classes d'équivalence + all pairs

- Les classes d'équivalence valides :

Classes d'équivalence « Jour »	Classes d'équivalence « Mois »	Classes d'équivalence « Année »	Jeux de valeur
Jour ∈ [1, 28]	Mois ∈ {1, 3, 5, 7, 8, 10}	Année bissextile	(14, 5, 2004)
Jour = 29	Mois = 12	Année non bissextile	(29, 12, 1999)
Jour = 30	Mois ∈ {4, 6, 9, 11}	Année = 2000	(30, 6, 2000)
Jour = 31	Mois = 2	Année bissextile	(31, 2, 1916)
Jour ∈ [1, 28]	Mois = 2	Année non bissextile	(15, 2, 1789)
Jour = 29	Mois ∈ {1, 3, 5, 7, 8, 10}	Année = 2000	(29, 12, 2000)
Jour = 30	Mois = 12	Année bissextile	(30, 12, 1992)
Jour = 31	Mois ∈ {4, 6, 9, 11}	Année non bissextile	(31, 9, 1995)
Jour ∈ [1, 28]	Mois ∈ {4, 6, 9, 11}	Année = 2000	(7, 11, 2000)
Jour = 29	Mois = 2	Année bissextile	(29, 2, 1964)
Jour = 30	Mois ∈ {1, 3, 5, 7, 8, 10}	Année non bissextile	(30, 7, 1903)
Jour = 31	Mois = 12	Année 2000	(31, 12, 2000)
Jour ∈ [1, 28]	Mois = 12	Année bissextile	(8, 12, 1888)
Jour = 29	Mois ∈ {4, 6, 9, 11}	Année non bissextile	(29, 9, 1805)
Jour = 30	Mois = 2	Année = 2000	(30, 2, 2000)
Jour = 31	Mois ∈ {1, 3, 5, 7, 8, 10}	Année bissextile	(31, 3, 2012)

La construction des classes d'équivalence : par l'exemple

La génération des jeux de tests : classes d'équivalence + all pairs

- Les classes d'équivalence invalides :

Jeux de valeurs	Classes d'équivalences couvertes
(-10, 2, 2000)	Jour < 1
(33, 12, 2001)	Jour > 31
(2, 0, 1999)	Mois < 1
(3, 14, 1997)	Mois > 12
(1, 1, 1500)	Année < 1582
(1, 1, 4000)	Année > 3000

Tester aux limites

- Le test aux limites consiste à choisir des valeurs qui sont aux frontières des domaines de fonctionnement du logiciel.
- Les frontières se définissent grâce aux domaines obtenus lors du calcul des classes d'équivalence :
  - par la spécification des variables d'entrées.
  - par la spécification des résultats.

Tester aux limites

Exemple : une fonction qui prends en paramètre le numéro de la wilaya et retourne son nombre d'habitants.

Validité	Classes d'équivalence	Représentants avec limites	Large couverture
Entrées valides	[1 – 48]	1, 30 , 48	1, 2, 30 , 47
Entrées invalides	[minInt – 1 [	-3000, 0	-3000, -1, 0
Entrées invalides	] 48 – MaxInt]	49, 1000	49 , 71 , 1000

Tester aux limites

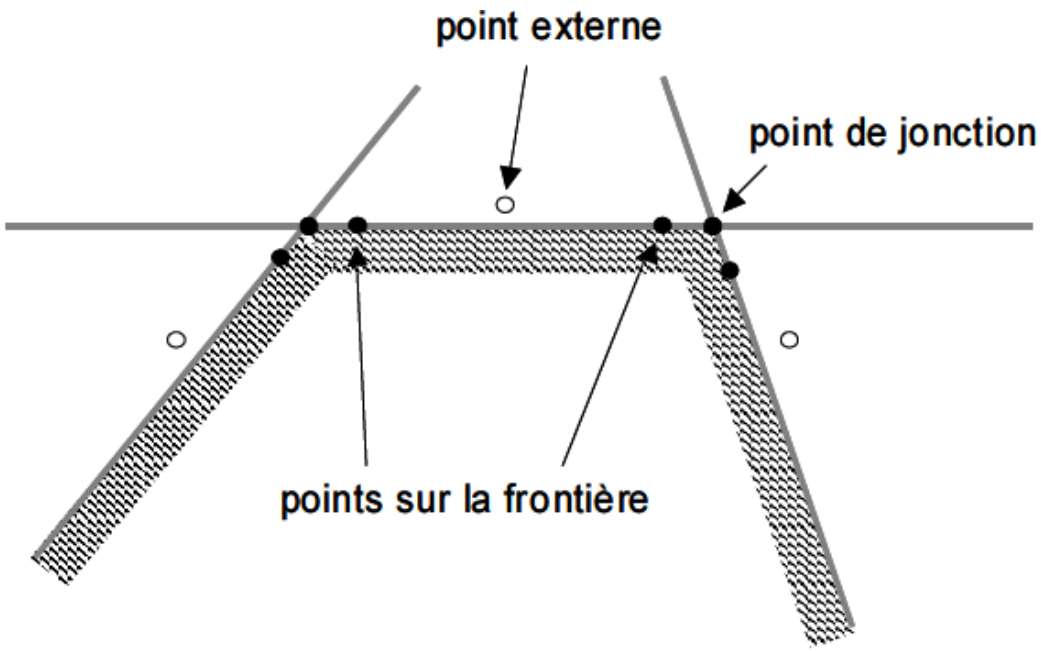
Exemple : une fonction qui prends **oui** ou **non** en paramètre.

Validité	Classes d'équivalence	Représentants avec limites
Entrées valides	{« oui »}	« oui »
Entrées valides	{« non »}	« non »
Entrées invalides	Autre chaîne	« hello word » « » « ouii » « mon »

Tester aux limites

Prendre en compte des contraintes liant les paramètres.

Exemple : un composant à deux paramètres X et Y, tel que ils vérifient : $(Y \prec 0)$ et $(X - Y \succ 0)$ et $(3X + Y - 15 \prec 0)$



Tester aux limites

On distingue deux formes de contraintes :

- Forme normale disjonctive : l'utilisation que du OU.
- Forme normale conjonctive : l'utilisation que du ET.

Tester aux limites

Si la condition est une conjonction (**ET**) de **M** prédicats booléens :

- Choisir un cas ou tous les prédicats sont à **VRAI**, c'est la partie *toutes les conditions* $\Longrightarrow$ l'expression vaut **VRAI** et la valeur attendue correspond à un calcul à partir d'entrée valide.
- Choisir **M** cas avec pour chacun un seul des prédicats **FAUX**, c'est la partie *chaque condition* $\Longrightarrow$ l'expression vaut **FAUX** et la valeur attendue correspond au traitement d'une entrée invalide.

Tester aux limites

Si la condition est une disjonction (**OU**) de **M** conditions booléennes :

- Choisir un cas ou tous les prédicats sont **FAUX**.
- Choisir M cas, avec pour chacun, un seul des prédicats à **VRAI**.

Tester aux limites

Exemple : un composant à deux paramètres X et Y, tel que ils vérifient : $(Y \prec 0)$ et $(X - Y \succ 0)$ et $(3X + Y - 15 \prec 0)$.

- Un cas ou les trois prédicats sont à **VRAI** : $X = 0$, $Y = -1$.
- Trois cas ou un seul prédicat est à **FAUX** :
  - $(Y \prec 0)$ FAUX et les autres à vrai : $X = 2$, $Y = 1$.
  - $(X - Y \succ 0)$ FAUX et les autres à vrai : $X = -2$, $Y = 1$.
  - $(3X + Y - 15 \prec 0)$ FAUX et les autres à vrai : $X = 6$, $Y = -1$.

Tester aux limites : par l'exemple

Tester une fonction *Lendemain* qui calcule le lendemain d'une date passée en paramètre et définie par trois entiers : jour, mois, année.

Classe d'équivalence	Valeurs limites possibles	Remarque
Jour $\in [1, 28]$	1 et 28	Bords de l'intervalle
Mois $\in \{1, 3, 5, 7, 8, 10\}$	1 ou 10	Première ou dernière valeur de l'énumération
Mois $\in \{4, 6, 9, 11\}$	4 ou 6	Même remarque que précédemment
Année bissextile	1600, 2400, 2000	Années « juste » bissextiles (quatre centenaires)
Année non bissextile	1582, 1700, 3000	Années « juste » non bissextiles (centenaires) ou limite de l'intervalle des valeurs possibles (1582 à 3000)

Tester aux limites : par l'exemple

Tester une fonction *Lendemain* qui calcule le lendemain d'une date passée en paramètre et définie par trois entiers : jour, mois, année.

Un compléments des jeux de tests

Classes d'équivalence « Jour »	Classes d'équivalence « Mois »	Classes d'équivalence « Année »	Jeux de valeurs aux limites
Jour $\in$ [1, 28]	Mois $\in$ {1, 3, 5, 7, 8, 10}	Année bissextile	(1, 1, 2400) (28, 10, 1600)
Jour $\in$ [1, 28]	Mois = 2	Année non bissextile	(28, 2, 1700)
Jour = 29	Mois = 2	Année bissextile	(29, 2, 1600) (29, 2, 2400) (29, 2, 2000)
Jour = 31	Mois = 12	Année 2000	(31, 12, 2000)
Jour = 31	Mois $\in$ {1, 3, 5, 7, 8, 10}	Année bissextile	(31, 3, 1600) (31, 12, 3000)

Les jeux de tests grace aux spécifications

- D'autres techniques existent :
  - Tester grâce à une table de décision.
  - Utiliser un diagramme états-transition.

Les jeux de tests grace aux spécifications

- D'autres techniques existent :
  - Tester grâce à une table de décision.
  - Utiliser un diagramme états-transition.
- Les jeux de tests obtenus grace aux spécifications sont utilisés dans les autres phases de tests :
  - Tests unitaires.
  - Tests systèmes.
  - Tests d'intégration.
  - ...