Tests et Qualité du code
Tests structurels (Boite blanche)

Youness LAGHOUAOUTA

Institut National des postes et télécommunications
laghouaouta@inpt.ac.ma

Février 2022

Typologie des tests

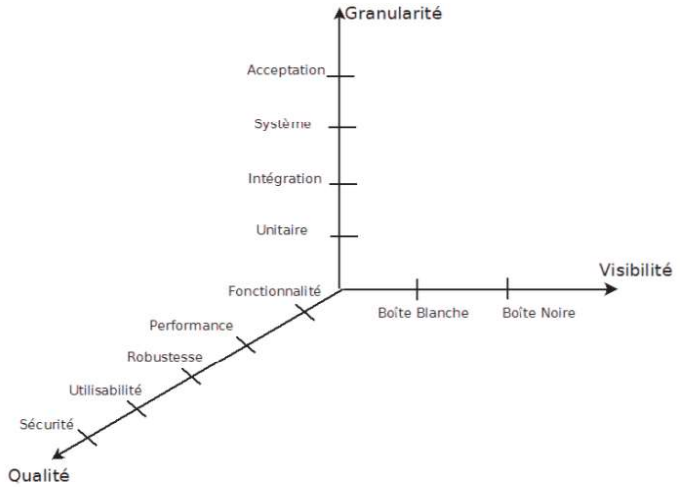
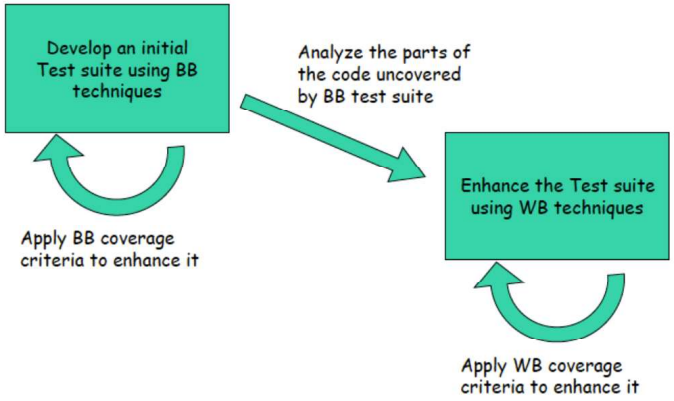


Figure: Typologie des tests [1]

Test structurel (Boite blanche)

- Sélection des tests à partir de l'analyse du code source du système
- Construction des tests uniquement pour du code déjà écrit
- Complémentarité entre les tests fonctionnels et structurels :
  - Test fonctionnel : détecte les oublis ou les erreurs par rapport à la spécification
  - Test structurel : détecte les erreurs de programmation

Complémentarité tests BB et BN



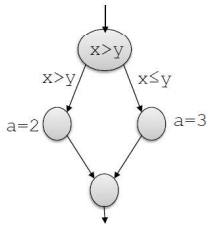
Sélection des tests

- Produire des DT qui couvriront un ensemble chemins d'exécution du programme
- Exhaustivité inenvisageable!! : nombre de chemins infini
- Critère d'arrêt = conditions objectives, mesurables, qui font qu'on peut considérer qu'une série de tests est suffisante
- Critère d'arrêt basé sur la couverture du code :
  - Flot de contrôle : analyse basée sur la couverture des éléments du code (ex. instructions, conditions, branches ...)
  - Flot de données : analyse fine des relations entre instructions en tenant compte des variables qu'elles utilisent/définissent
  - ...
- Utilisation importante des parcours de graphes
- Approprié pour le test unitaire ou d'intégration, mais il passe mal à l'échelle

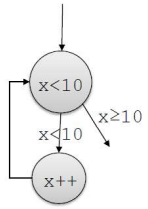
Graphe de flot de contrôle

- Un graphe de flot de contrôle est une représentation abstraite et simplifiée de la structure du programme
  - Décrit le flot de contrôle du programme
  - Chaque nœud représente un segment de code strictement séquentiel (sans choix ou boucle)
  - Chaque arc dénote un lien de contrôle (possibilité d'aller d'un nœud à l'autre)
- Graphe orienté avec un nœud initial E et un nœud final S tel que :
  - Un nœud interne est :
    - Soit un bloc d'instructions élémentaires
    - Soit un nœud de décision étiqueté par une condition
  - Un arc relie les nœuds correspondant à des instructions ou conditions successives (flot de contrôle)

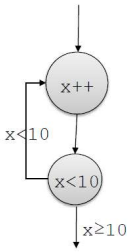
Graphe de flot de contrôle



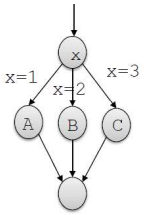

```
if (x>y)
  a = 2;
else
  a = 3;
```




```
while(x<10)
     x++;
```




```
do
   x++;
while (x<10)
```



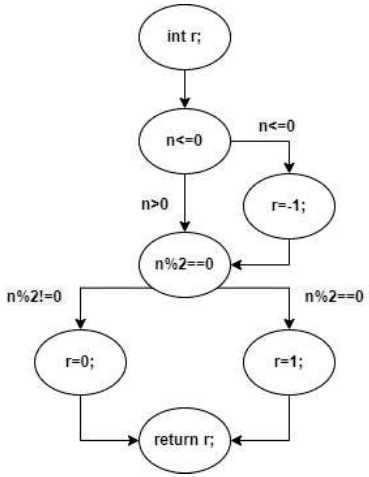

```
switch(x) {
case 1:
     A;break;
case 2:
     B;break;
case 3:
     C;break;
}
```


Exemple 1

```
int fonction(int n){
   int r;
   if(n<=0){
        r=-1;
   }
   if(n%2==0){
        r=1;
   }
   else{
        r=0;
   }
   return r;
}
```

Exemple 1

```
int fonction(int n){
   int r;
   if(n<=0){
        r=-1;
   }
   if(n%2==0){
        r=1;
   }
   else{
        r=0;
   }
   return r;
}
```

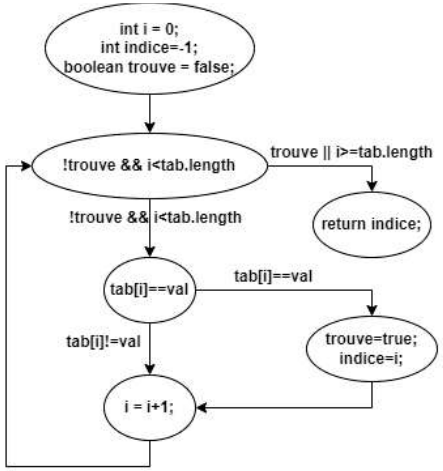


Exemple 2

```
int indexOf(int[] tab, int val){
  int i = 0;
  int indice=-1;
  boolean trouve = false;
  while(!trouve && i<tab.length){
        if(tab[i]==val){
                trouve=true;
                indice=i;
        }
        i = i+1;
  }
  return indice;
}
```


Exemple 2

```
int indexOf(int[] tab, int val){
  int i = 0;
  int indice=-1;
  boolean trouve = false;
  while(!trouve && i<tab.length){
        if(tab[i]==val){
                trouve=true;
                indice=i;
        }
        i = i+1;
  }
  return indice;
}
```



Couverture du CFG

- Tous les nœuds : le plus faible
  - Couverture des instructions
- Tous les arcs : test de chaque décision
  - Couverture des branchements et conditions
- Tous les chemins : le plus fort, impossible à réaliser s'il y a des boucles
  - Couverture exhaustive
  - Certains chemins peuvent ne pas être atteignables

Tests pour un critère de couverture

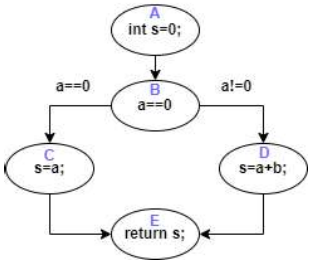
1. Sélectionner un ensemble (minimal) de chemins satisfaisant le critère
2. Pour chaque chemin, définir une donnée de test (DT) permettant de le sensibiliser
   - Si un chemin est sensibilisé par une DT, il est dit exécutable
   - Si aucune DT ne permet de sensibiliser un chemin, il est dit non exécutable

Tous les nœuds

```
int somme(int a, int b){
   int s=0;
   if(a==0)
        s=a;
   else
        s=a+b;
   return s;
}
```


Tous les nœuds

```
int somme(int a, int b){
   int s=0;
   if(a==0)
        s=a;
   else
        s=a+b;
   return s;
}
```



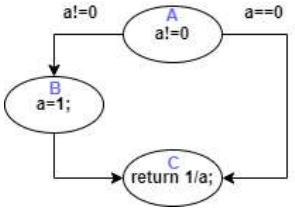
- Chemins de contrôle pour couvrir tous les nœuds :
  1. ABDE sensibilisé avec la DT $\{a = 2; b = 1\}$
  2. ABCE sensibilisé avec la DT $\{a = 0; b = 2\}$
- Détection de l'erreur avec le deuxième chemin

Tous les nœuds

```
int inverse(int a){
   if(a!=0)
        a=1;
   return 1/a;
}
```

Tous les nœuds

```
int inverse(int a){
   if(a!=0)
        a=1;
   return 1/a;
}
```



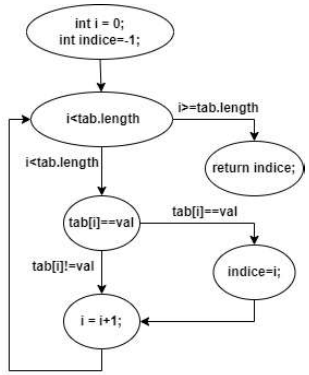
- Le chemin de contrôle ABC couvre tous les nœuds et il sensibilisé par la DT $\{a = 1\}$
- Pas de détection de l'erreur (division par 0)

Tous les arcs

```
int indexOf(int[] tab, int val){
   int i=0;
   int indice=-1;
   while(i<tab.length){
        if(tab[i]==val){
           indice=i;
        }
        i=i+1;
   }
   return indice;
}
```

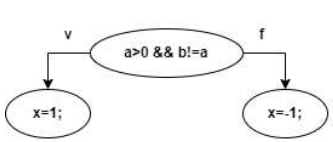

Tous les arcs

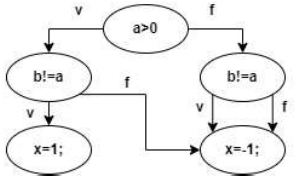
```
int indexOf(int[] tab, int val){
   int i=0;
   int indice=-1;
   while(i<tab.length){
        if(tab[i]==val){
           indice=i;
        }
        i=i+1;
   }
   return indice;
}
```



- La DT $\{tab = \{9, 1, 2, 1, 6\}, val = 2\}$ permet de couvrir tous les arcs
- Mais c'est la DT $\{tab = \{9, 1, 2, 1, 6\}, val = 1\}$ qui met le programme en erreur

Problème des conditions multiples





- Décomposition des conditions
- Couverture de toutes les conditions multiples

Nombre de chemins exponentiel en fonction du nombre de sous-conditions

Toutes les conditions Multiples

if(A && B && C)

- Toutes les décisions :

A	B	C	Décision
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Toutes les conditions Multiples

if(A && B && C)

- Toutes les décisions :

A	B	C	Décision
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Toutes les conditions Multiples

if(A && B && C)

- Toutes les conditions multiples :

A	B	C	Décision
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Toutes les conditions Multiples

if(A && B && C)

- Toutes les conditions multiples :

A	B	C	Décision
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

MC/DC

MC/DC : modified condition/decision coverage

- Objectif : améliorer le critère de couverture toutes les décisions en contrôlant la combinatoire
- On ne considère une combinaison de valeurs faisant varier une sous-condition que si cette sous-condition influence la décision
- A pour but de démontrer l'action de chaque condition sur la valeur de vérité de l'expression (appelée décision)

MC/DC

if(A && B && C)

- Toutes les conditions/décisions modifiées :

A	B	C	Décision
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

MC/DC

if(A && B && C)

- Toutes les conditions/décisions modifiées :

A	B	C	Décision
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

MC/DC Exemple2

if(A && (B || C))

- Toutes les conditions/décisions modifiées :

A	B	C	Décision
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

MC/DC Exemple2

if(A && (B || C))

- Toutes les conditions/décisions modifiées :

A	B	C	Décision
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Tous les chemins

- Il est impossible de couvrir tous les chemins en présence de boucles
- Une seule boucle (simple), tester les exécutions :
  - Aucune itération
  - Une seule itération
  - Deux itérations
  - Un nombre typique d'itérations
  - n 1, n, n + 1 itérations avec n le nombre d'itérations.
- Boucles imbriquées : commencer par fixer le nombre d'itérations de la boucle la plus extérieure et tester les boucles intérieures comme boucle simple.
- Boucles en suite : si dépendantes, tester comme imbriquées, sinon tester comme simple

Graphe de Flot de Données

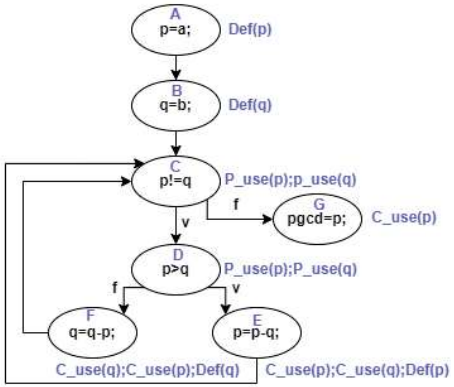
- Graphe de contrôle décoré d'informations sur les données (variables) du programme
- Annotation du graphe de flot de contrôle par les définitions et les utilisations de variables
  - une définition (= affectation) d'une variable v est notée Def(v)
  - une utilisation d'une variable est v notée P_use(v) dans un prédicat et C_use(v) dans un calcul .
- Base pour l'analyse des dépendances entre les définitions et utilisations d'une même variable : ordre des annotations important

Exemple

```
p=a;
q=b;
while(p!=q){
        if(p>q){
                p=p-q;
        }
        else{
                q=q-p;
        }
}
pgcd=p;
```

Exemple

```
p=a;
q=b;
while(p!=q){
        if(p>q){
                p=p-q;
        }
        else{
                q=q-p;
        }
}
pgcd=p;
```



- Pour le chemin ABCG:
  - Flot de données pour p : Def(p)->P_use(p)->C_use(p)

Analyse du flot de données

- Analyse des dépendances entre les définitions et utilisations d'une variable
- Flot de données pour une variable X :
  - Pour une exécution du programme : suite des définitions et utilisations
  - Pour toutes les exécutions du programme : ensemble des suites de définitions et utilisations

Anomalies du flot de données :

- (P|C)_use(X)...: variable non initialisée
- ...Def(X) : mise à jour jamais utilisée
- ...Def(X)Def(X) ...: mise à jour non utilisée

Réferences

1 Blasquez, Isabelle, Hervé Leblanc, and Christian Percebois. "Les tests dans le développement logiciel, du cycle en V aux méthodes agiles." Revue des Sciences et Technologies de l'Information-Série TSI: Technique et Science Informatiques 36.1-2 (2017): 7-50.

2 IEEE standard glossary of software engineering terminology. IEEE Std 610.12-1990.