Test des logiciels

Remarque : Ce cours est la suite du chapitre 1 (Introduction au Génie Logiciel)

1. Introduction

Le test est un processus manuel ou automatique, qui vise à établir qu'un système vérifie les propriétés exigées par sa spécification, ou à détecter des différences entre les résultats engendrés par le système et ceux qui sont attendus par la spécification.

- Le test vise à mettre en évidence les erreurs d'un logiciel ;
- Le test n'a pas pour objectif de corriger les erreurs ;
- Le test n'a pas pour objectif de prouver la correction d'un programme.

2. Types de test

Il existe plusieurs types de tests :

- **Test unitaire** : consiste à vérifier chaque composant indépendamment des autres. On distingue :
  - Test fonctionnel : examine les contraintes liées aux spécifications et évaluent les réactions du logiciel à certaines entrées, sans rentrer à l'intérieur des modules.
  - Test structurel : examine la structure de chaque module via le graphe de flot de contrôle[1]; le code du programme est testé par partie (on parle de portions) et on nommera, à la fin de cette série de tests, « couverture » l'ensemble des portions de code qui auront été testées.
- **Test de sous-système** : les composants pouvant communiquer les uns avec les autres, il s'agit donc de vérifier les interfaces entre composants.
- **Test d'intégration** : consiste à tester un ensemble de composants qui viennent d'être assemblés.
- **Test d'acceptation :** établi avec la participation du client pour obtenir son acceptation.

3. Méthodes de tests

On peut distinguer deux méthodes de test :

3.1. Méthodes de tests statiques

Ces méthodes consistent en l'analyse textuelle du code du logiciel afin d'y détecter des erreurs, sans exécution du programme.

Avantages :

- méthodes efficaces et peu coûteuses ;
- 60% à 95% des erreurs sont détectées lors de contrôles statiques.

Inconvénients :

- méthodes ne permettant pas de valider le comportement d'un programme au cours de son exécution ;
- lors d'une modification du programme, il est souvent difficile de réutiliser les tests précédents pour valider la nouvelle version ;
- les méthodes de tests statiques ne sont pas suffisantes.

[1]Le graphe de flot de contrôle (le graphe de contrôle) permet de représenter n'importe quel algorithme.

3.2. Méthodes de tests dynamiques

Les méthodes de tests dynamiques consistent en l'exécution du programme à valider à l'aide d'un jeu de tests. Elles visent à détecter des erreurs en confrontant les résultats obtenus par l'exécution du programme à ceux attendus par la spécification de l'application.

4. La construction des jeux de tests

On distingue l'approche aléatoire, l'approche fonctionnelle ou « boîte noire » et l'approche structurelle ou « boîte blanche ».

a) L'approche aléatoire :

Le jeu de tests est sélectionné au hasard sur le domaine de définition des entrées du programme. Cette méthode ne garantit pas une bonne couverture de l'ensemble des entrées du programme. En particulier, elle peut ne pas prendre en compte certains cas limites ou exceptionnels. Cette méthode a donc une efficacité très variable.

b) L'approche fonctionnelle ou boîte noire *:*

On considère seulement la spécification de ce que doit faire le programme, sans considérer sa structure interne. On peut vérifier chaque fonctionnalité décrite dans la spécification. On s'appuie principalement sur les données et les résultats. Le danger est l'explosion combinatoire qu'entraîne un grand nombre d'entrées du programme. Par contre, on peut écrire ces tests très tôt, dès qu'on connaît la spécification.

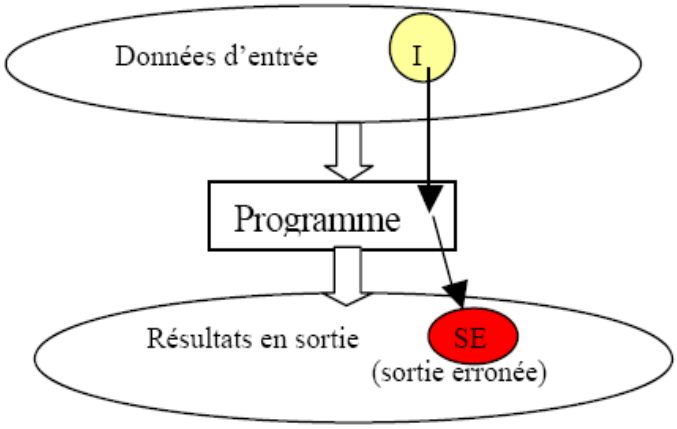

Figure 1 : Tests boîte noire

Une technique souvent utilisée est l'analyse des valeurs frontières et le regroupement en classes d'équivalence. Elle se base sur l'observation que les erreurs arrivent souvent aux limites des domaines de définition des variables du programme. Au lieu de tester toutes les valeurs, on partitionne les valeurs en classes d'équivalence et on teste aux limites des classes.

Exemple : si une donnée doit varier dans un intervalle [a,b], on peut définir 3 classes < a, dans l'intervalle et > b.

c) L'approche structurelle ou boîte blanche :
Dans l'approche par boîte blanche on tient compte de la structure interne du module. On peut s'appuyer sur différents critères pour conduire le test comme :
c1) Le critère de couverture des instructions : le jeu d'essai doit assurer que toute instruction élémentaire est exécutée au moins une fois.
c2) Le critère de couverture des arcs du graphe de contrôle : le graphe de contrôle est un graphe qui résume les structures de contrôle d'un programme.

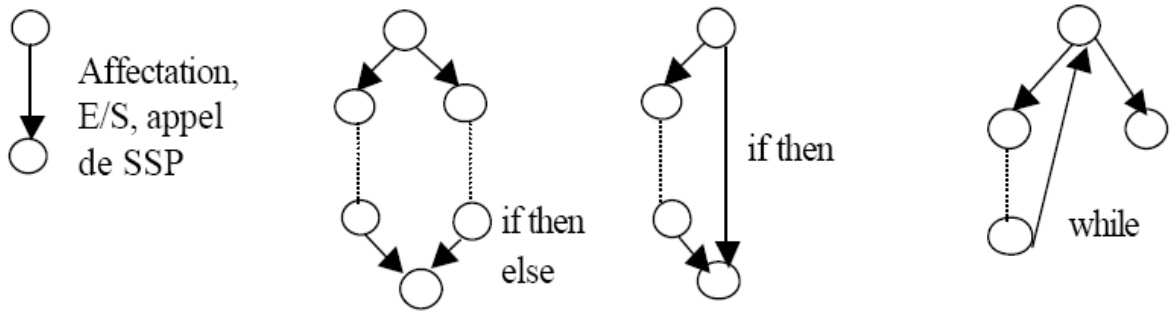

Figure 2 : Eléments d'un graphe de contrôle

Exemple 1 : soit l'algorithme d'Euclide qui calcule le pgcd de 2 nombres (plus grand commun diviseur) et son graphe de contrôle.

```
begin
read(x); read(y);
while (not (x = y)) loop
if x > y then
x := x - y;
else
y := y - x;
end if;
end loop;
pgcd := x;
end;
```

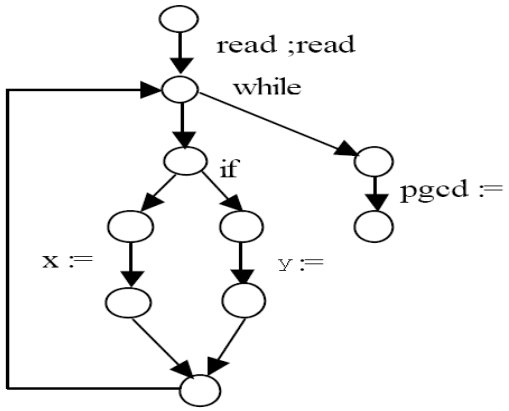

Figure 3 : Graphe de contrôle de l'algorithme d'Euclide

Le jeu d'essai (x=3, y=3), (x=4, y=3), (x=3, y=4) satisfait les 2 critères de couverture.

Exemple 2 : soit le programme suivant

```
if x < 0 then
x := -x;
end if;
z := x;
```

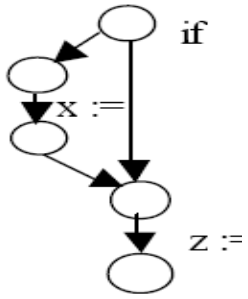

Figure 3 : Graphe de contrôle d'un if sans else

(x=-2) satisfait le critère de couverture des instructions mais pas celui des arcs. Il faudrait aussi tester ce qui se passe quand x est positif.

c3) Le critère de couverture des chemins du graphe de contrôle

Exemple 3 : soit le programme suivant

```
if (not (x=0)) then
y := 5;
else
z := z - x;
end if;
if (z > 1) then
z := z / x;
else
z := 0;
end if;
```

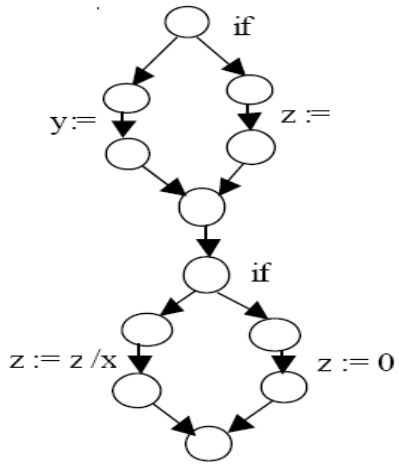

Figure 4 : Graphe de contrôle d'un double conditionnel

Le jeu d'essai (x=0, z=1), (x=1, z=3) vérifie la couverture des arcs mais pas celle des chemins et ne peut donc détecter une division par zéro.
(x=0, z=3), (x=1, z=1), (x=0, z=1), (x=1, z=3) vérifie la couverture des chemins et détecte donc la division par zéro. Malheureusement le nombre de chemins peut devenir très grand dans des programmes réels ce qui rend ce dernier critère souvent inexploitable en pratique.

c4) Le critère de couverture des conditions : le jeu de tests doit couvrir à vrai et à faux toutes les conditions élémentaires de toutes les conditionnelles.

Exemple 4:

```
if (a > 1 and b = 0) then     a > 1    False      True
x := x / a;                   b = 0    False      True
end if;                       a = 2    False      True
if (a = 2 or x > 1)  then     x > 1    False      True
x := x + 1;
end if;
```

(a=1, b=0, x=3), (a=2, b=1, x=1) couvre toutes les conditions mais pas toutes les instructions.

Remarque : En général il est conseillé de mélanger différents critères.