Introduction au test logiciel

Fabrice AMBERT, Fabrice BOUQUET, Fabien PEUREUX,
Jean-Marie GAUTHIER, Alexandre VERNOTTE
prenom.nom@femto-st.fr

ENVOL 2014
La Londe-les-Maures
18-21 novembre 2014

Agenda

- Mercredi 19/11/14 : Test structurel
- Jeudi 20/11/14 : Test fonctionnel

Au menu :

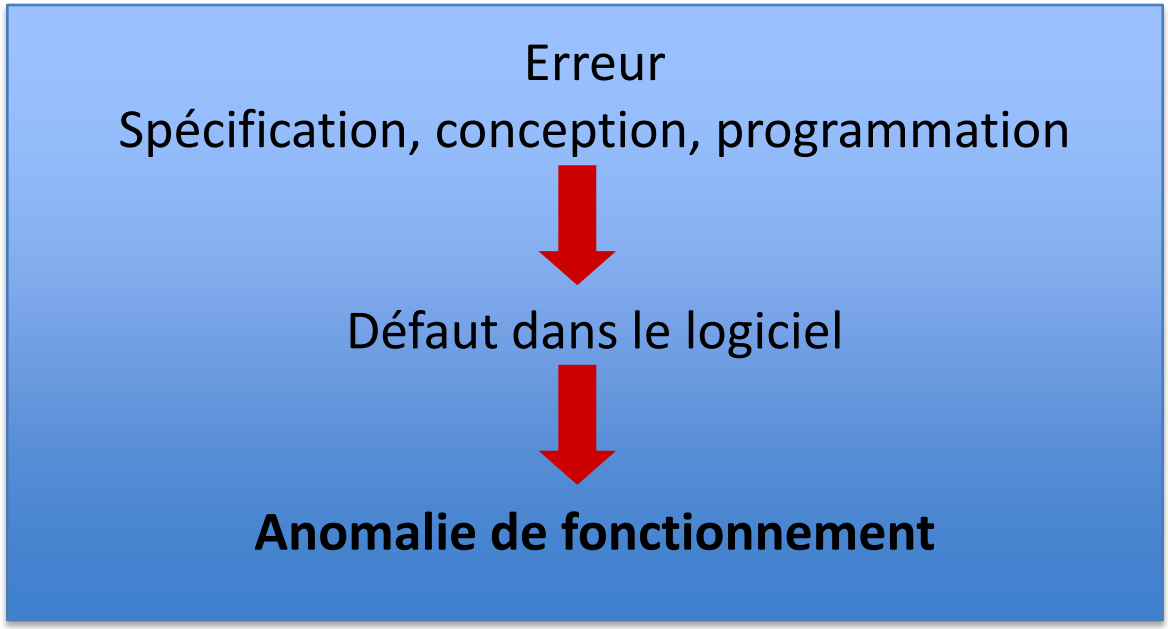
9h00 – 10h30 : Introduction des concepts

11h00 – 12h30 : Exercice d'introduction

13h30 – 19h00 : Mise en œuvre pratique

Motivations du test

- Complexité croissante des architectures et des comportements
- Coût d'un bug (Ariane 5, carte à puces allemande bloquée, prime de la PAC…)



- Coût des bugs informatiques : ≈ 60 milliards $ / an
- 22 milliards économisés si les procédures de test de logiciels étaient améliorées.

(source : NIST - National Institute of Standards and Technology - 2002)

Validation & Vérification

- V & V
  - Validation : Est-ce que le logiciel offre les services attendues ?
  - Vérification : Est-ce que les artefacts utilisés sont corrects ?

- Méthodes de V & V
  - Test statique : Revue de code, de spécifications, de documents de design
  - Test dynamique : Exécuter le code pour s'assurer d'un fonctionnement correct
  - Vérification symbolique : Run-time checking, Execution symbolique, …
  - Vérification formelle : Preuve ou model-checking d'un modèle formel

Définition du test

- *« Le test est l'exécution ou l'évaluation d'un système ou d'un composant par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus. »*
  IEEE (Standard Glossary of Software Engineering Terminology)

- *« Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts. »*
  G. Myers (The Art of Software testing)

- *«Testing can reveal the presence of errors but never their absence.»*
  Edsger W. Dijkstra (Notes on Structured Programming)

La pratique du test (1)

- Le test appartient à l'activité de Validation du logiciel :
  est-ce-que le logiciel fait les choses bien et les bonnes choses ?

- Activité historiquement peu populaire en entreprise

- Difficultés d'ordre psychologique ou « culturel » :
  - L'activité de programmation est un processus constructif : on cherche à établir des résultats corrects
  - Le test est un processus destructif : un bon test est un test qui trouve une erreur

Méthodes de test

- **Test statique**
  Traite le code du logiciel sans l'exécuter sur des données réelles.

- **Test dynamique**
  Repose sur l'exécution effective du logiciel pour un sous ensemble bien choisi du domaine de ses entrées possibles.

La réalité du test

- Le test constitue aujourd'hui le vecteur principal de l'amélioration de la qualité du logiciel

- Actuellement, le test dynamique est la méthode la plus diffusée
- Il peut représenter jusqu'à 60 % de l'effort complet de développement d'un logiciel

- Coût moyen de l'activité de test :
  - 1/3 durant le développement du logiciel
  - 2/3 durant la maintenance du logiciel

Test statique

- Exemples :
  - *Lectures croisées / inspections*
    Vérification collégiale d'un document (programme ou spécification du logiciel)
  - *Analyse d'anomalies*
    Corriger de manière statique les erreurs (typage impropre, code mort, …)

- Avantages :
  - Méthodes efficaces et peu coûteuses
  - 60% à 95% des erreurs sont détectées lors de contrôles statiques

- Inconvénients :
  - Ne permet pas de valider le comportement d'un programme au cours de son exécution

→ Les méthodes de test statiques sont nécessaires, mais pas suffisantes

Test dynamique - niveaux

- Repose sur l'exécution du programme à tester

- 4 niveaux complémentaires :
  - Test de composants unitaires (structurel)
  - Test d'intégration des composants
  - Test du système global (fonctionnel)
  - Test d'acceptation (recette)

Test dynamique - techniques

- Deux techniques :
  - **Test structurel**
    Jeu de test sélectionné en s'appuyant sur une analyse du code source du logiciel (*test boite blanche / boite de verre*)
  - **Test fonctionnel**
    Jeu de test sélectionné en s'appuyant sur les spécifications (*test boite noire*)
- En résumé, les méthodes de test dynamique consistent à :
  - Exécuter le programme sur un ensemble fini de données d'entrées
  - Contrôler la correction des valeurs de sortie en fonction de ce qui est attendu

Test de logiciels – auto-évaluation
L'exemple du triangle

- Soit la spécification suivante :

  Un programme prend en entrée trois entiers. Ces trois entiers sont interprétés comme représentant les longueurs des cotés d'un triangle. Le programme rend un résultat précisant s'il s'agit d'un triangle scalène, isocèle ou équilatéral (ou une erreur si les données ne correspondent pas aux longueurs d'un triangle).

  G. Myers (The Art of Software testing - 1979)

- Donnez un ensemble de cas de test que vous pensez adéquat pour tester pour ce programme…

Test de logiciels – auto-évaluation
L'exemple du triangle

- Avez-vous un cas de test pour :
  1. Cas scalène valide (1,2,3 et 2,5,10 ne sont pas valides)
  2. Cas équilatéral valide
  3. Cas isocèle valide (2,2,4 n'est pas valide)
  4. Cas isocèle valide avec les trois permutations (e.g. 3,3,4; 3,4,3; 4,3,3)
  5. Cas avec une valeur à 0
  6. Cas avec une valeur négative
  7. Cas où la somme de deux entrées est égale à la troisième entrée
  8. Trois cas pour le test 7 avec les trois permutations
  9. Cas où la somme de deux entrées est inférieure à la troisième entrée
  10. Trois cas pour le test 9 avec les trois permutations
  11. Cas avec les trois entrées à 0
  12. Cas avec une entrée non entière
  13. Cas avec un nombre erroné de valeurs (e.g. 2 entrées, ou 4)
  14. Pour chaque cas de test, avez-vous défini le résultat attendu ?

Test de logiciels – auto-évaluation
L'exemple du triangle

- Chacun de ces 14 tests correspond à un défaut constaté dans des implantations de cet exemple du triangle

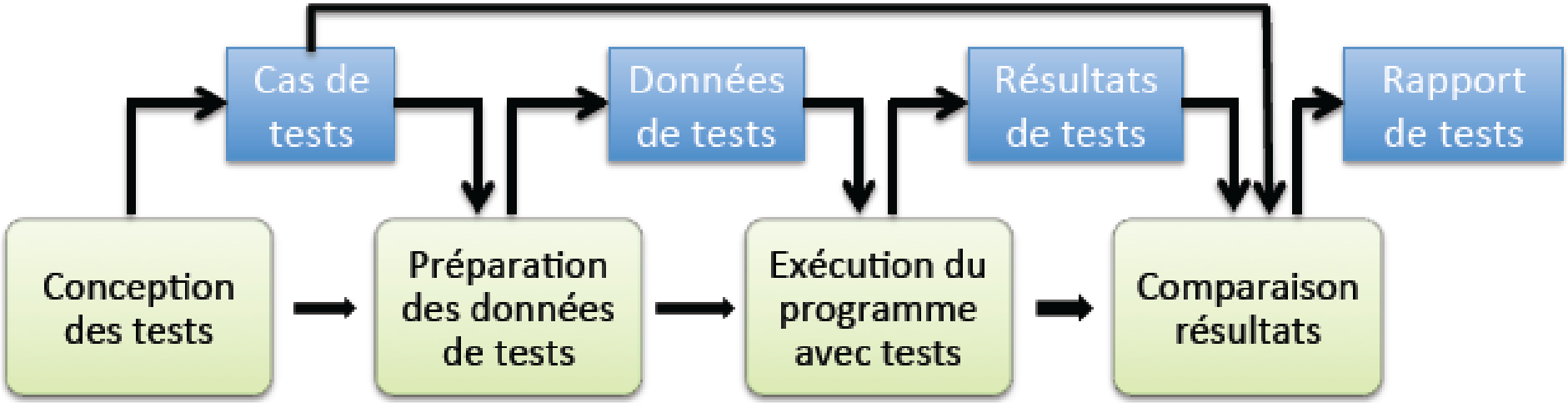
- La moyenne des résultats obtenus par un ensemble de développeurs expérimentés est de 7.8 sur 14.

=> Le test est une activité complexe, à fortiori sur de grandes applications

Test dynamique – 4 activités

- Sélection d'un jeu de tests : choisir un sous-ensemble des entrées possibles du logiciel
- Exécution du jeu de tests
- Dépouillement des résultats : consiste à décider du succès ou de l'échec du jeu de test (*verdict*):
  - Fail, Pass, Inconclusive
- Évaluation de la qualité et de la pertinence des tests effectués (déterminant pour la décision d'arrêt de la phase de test)

Test et cycle de vie du processus de test



Test structurel (white box)

- Les données de test sont produites à partir d'une analyse du code source

Critères de test :
- tous les chemins,
- toutes les instructions,
- etc…

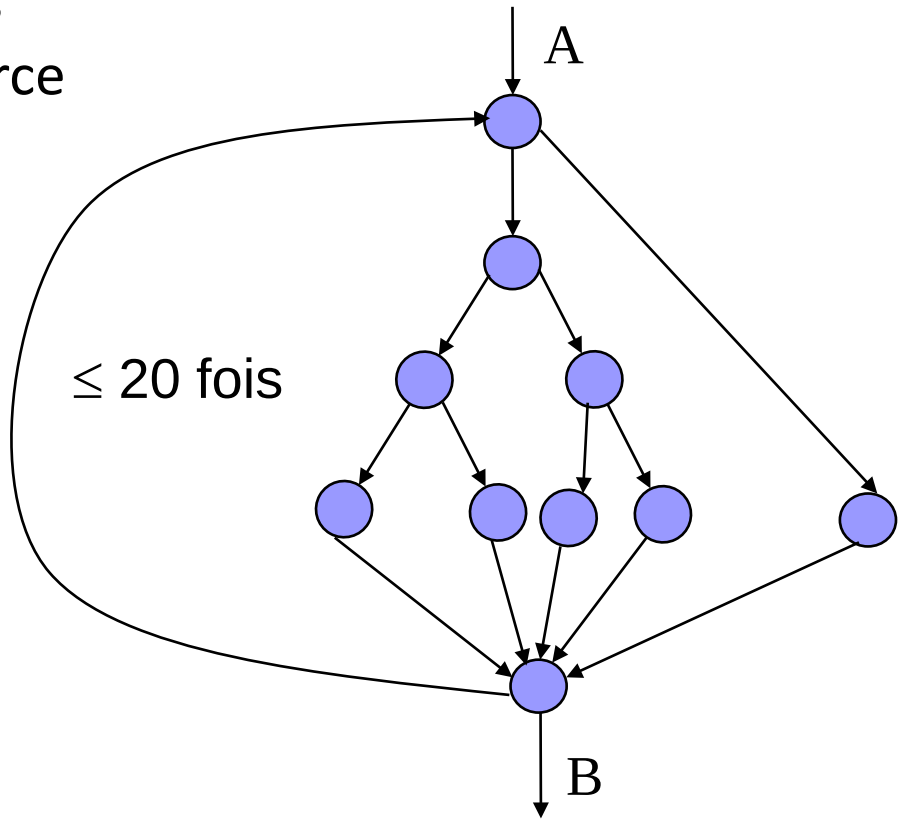
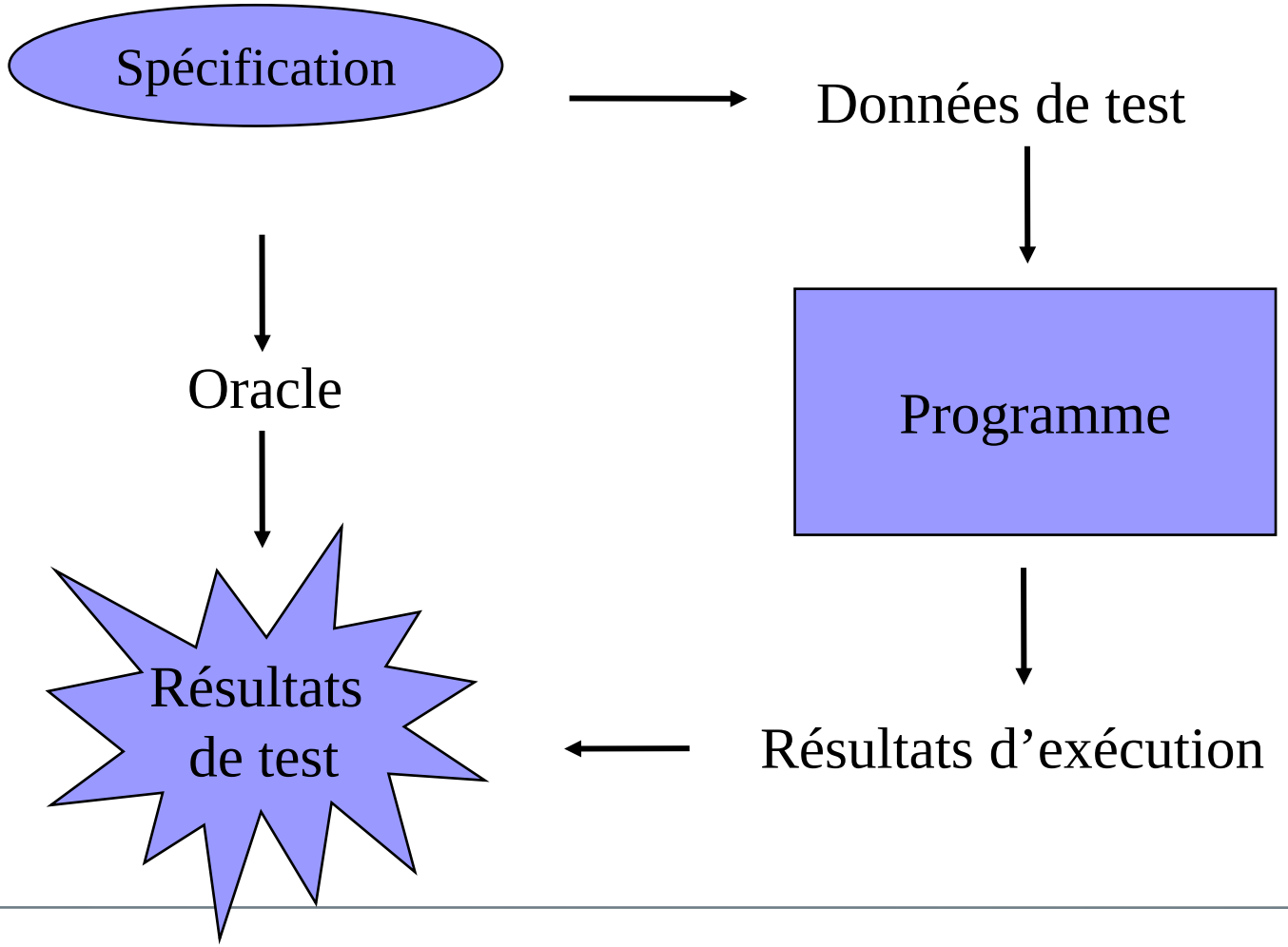


Fig. 1 : Flot de contrôle d'un petit programme

Test fonctionnel (black-box)

- Test de conformité par rapport à la spécification



Complémentarité (1)
test fonctionnel / structurel

- Les 2 approches sont utilisées de façon complémentaire

- Exemple : soit le programme suivant, censé calculer la somme de 2 entiers :

```
function sum (x,y : integer) : integer;
    if (x = 600) and (y = 500)
    then
        sum := x-y
    else
        sum := x+y;
    end
```

- Une approche fonctionnelle détectera difficilement le défaut, alors qu'une approche par analyse de code pourra produire la donnée de test : x = 600, y = 500.

Complémentarité (2) test fonctionnel / structurel

- En examinant ce qui a été réalisé, on ne prend pas forcément en compte ce qui aurait du être fait :
  - Les approches structurelles détectent plus facilement les erreurs commises dans le programme
  - Les approches fonctionnelles détectent plus facilement les erreurs d'omission et de spécification

- Une difficulté du test structurel consiste dans la définition de l'oracle de test.

Difficultés du test

- Le test exhaustif est en général impossible à réaliser
  - En test structurel, le parcours du graphe de flot de contrôle conduit à une forte explosion combinatoire

    Exemple : le nombre de chemin logique dans le graphe de la figure 1 est supérieur à $10^{14} \approx 5^{20} + 5^{19} + \ldots + 5^{1}$

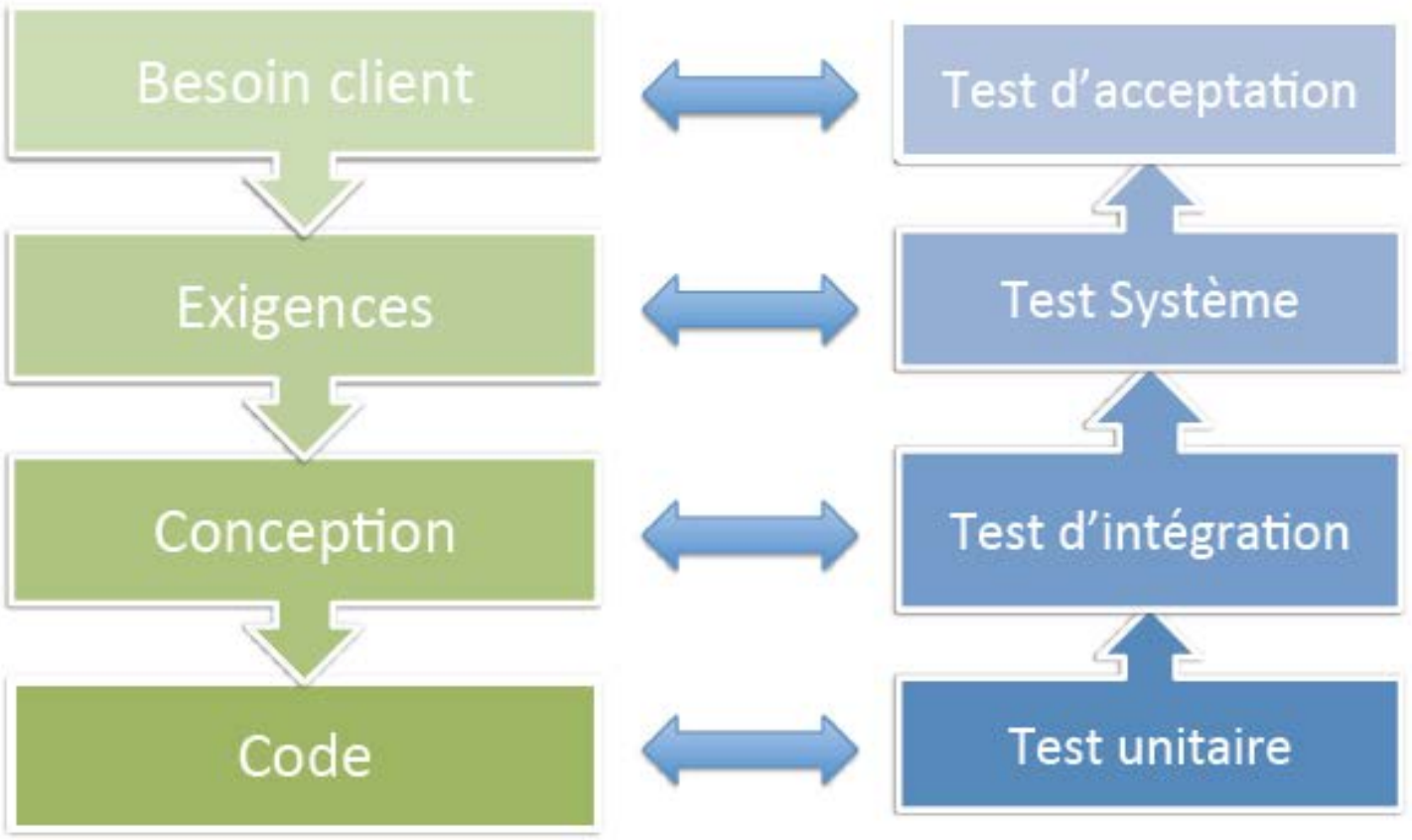
  - En test fonctionnel, l'ensemble des données d'entrée est en général infini ou très grande taille

    Exemple : un logiciel avec 5 entrées analogiques sur 8 bits admet 2^{40} *valeurs différentes en entrée*

=> le test est une méthode de validation partielle de logiciels

=> la qualité du test dépend de la pertinence du choix des données de test

Développement et niveaux de tests



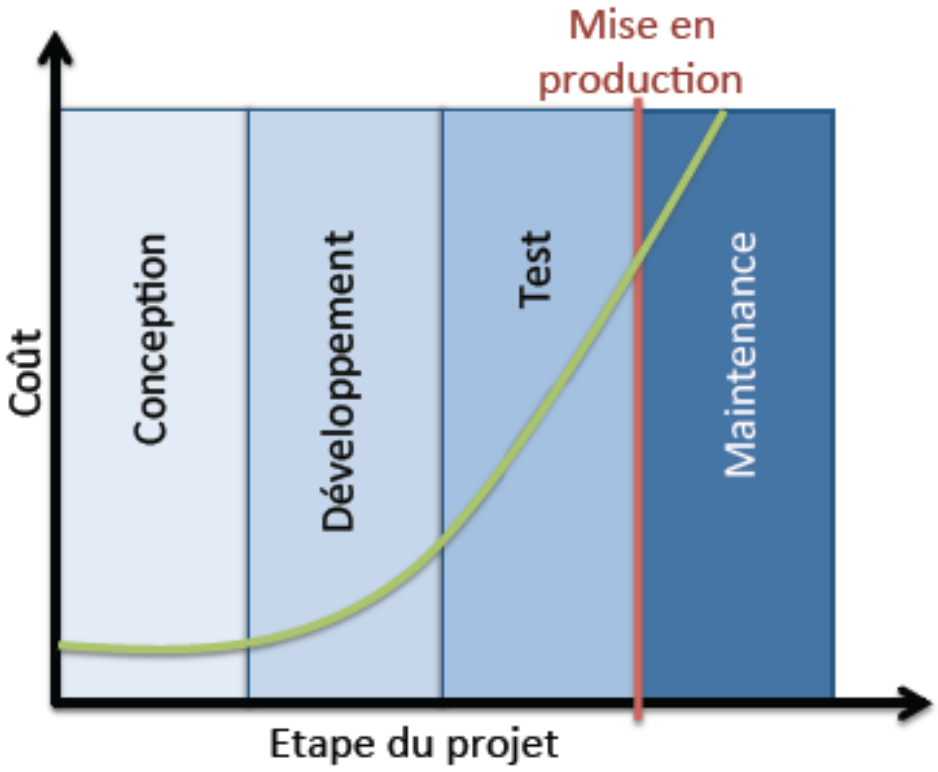
Niveaux de tests (renseignent l'objet du test)

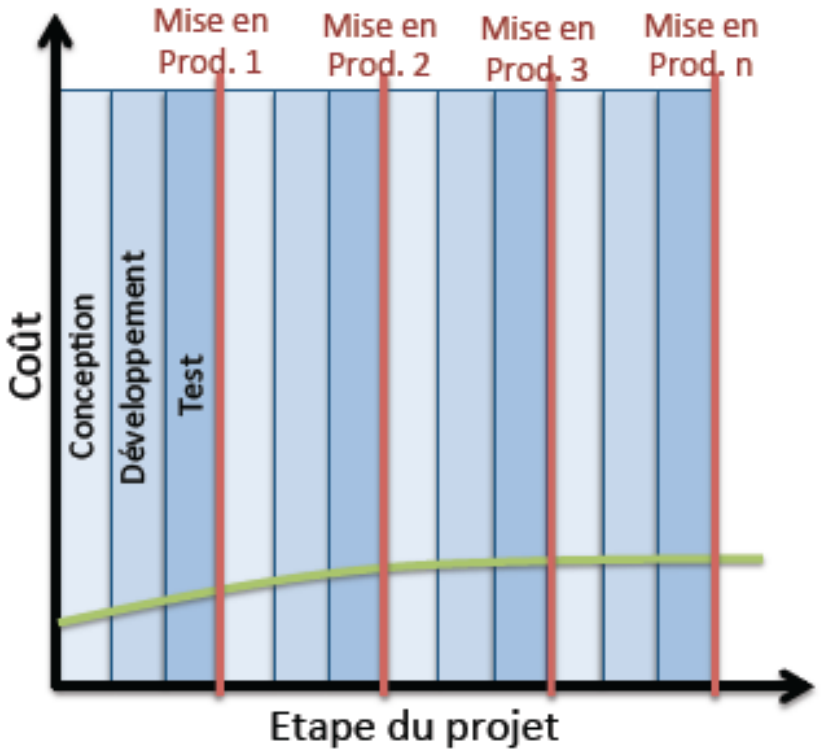
- **Tests (structurels) unitaires**
  Test de procédures, de modules, de composants
  (coût : 50% du coût total du développement initial correspondant)

- **Tests d'intégration**
  Test de bon comportement lors de la composition des procédures, modules ou composants
  (coût d'un bug dans cette phase : 10 fois celui d'un bug unitaire)

- **Tests système / de recette**
  Validation de l'adéquation aux spécifications
  (coût d'un bug dans cette phase : 100 fois celui d'un bug unitaire)

Coût d'un bug


Mise en production
Coût
Conception
Développement
Test
Maintenance
Etape du projet


Mise en Prod. 1
Mise en Prod. 2
Mise en Prod. 3
Mise en Prod. n
Coût
Conception
Développement
Test
Etape du projet

Types de tests (1)
(renseignent la nature du test mené)

- **Tests fonctionnels**

  Valide les résultats rendus par les services

- **Tests non-fonctionnels**

  Valide la manière dont les services sont rendus

- **Tests nominaux / de bon fonctionnement**
  Vérifie que le résultat calculé est le résultat attendu, en entrant des données valides au programme (*test-to-pass*)

- **Tests de robustesse**
  Vérifie que le programme réagit correctement à une utilisation non conforme, en entrant des données invalides (*test-to-fail*)

Types de tests (2)
(renseignent la nature du test mené)

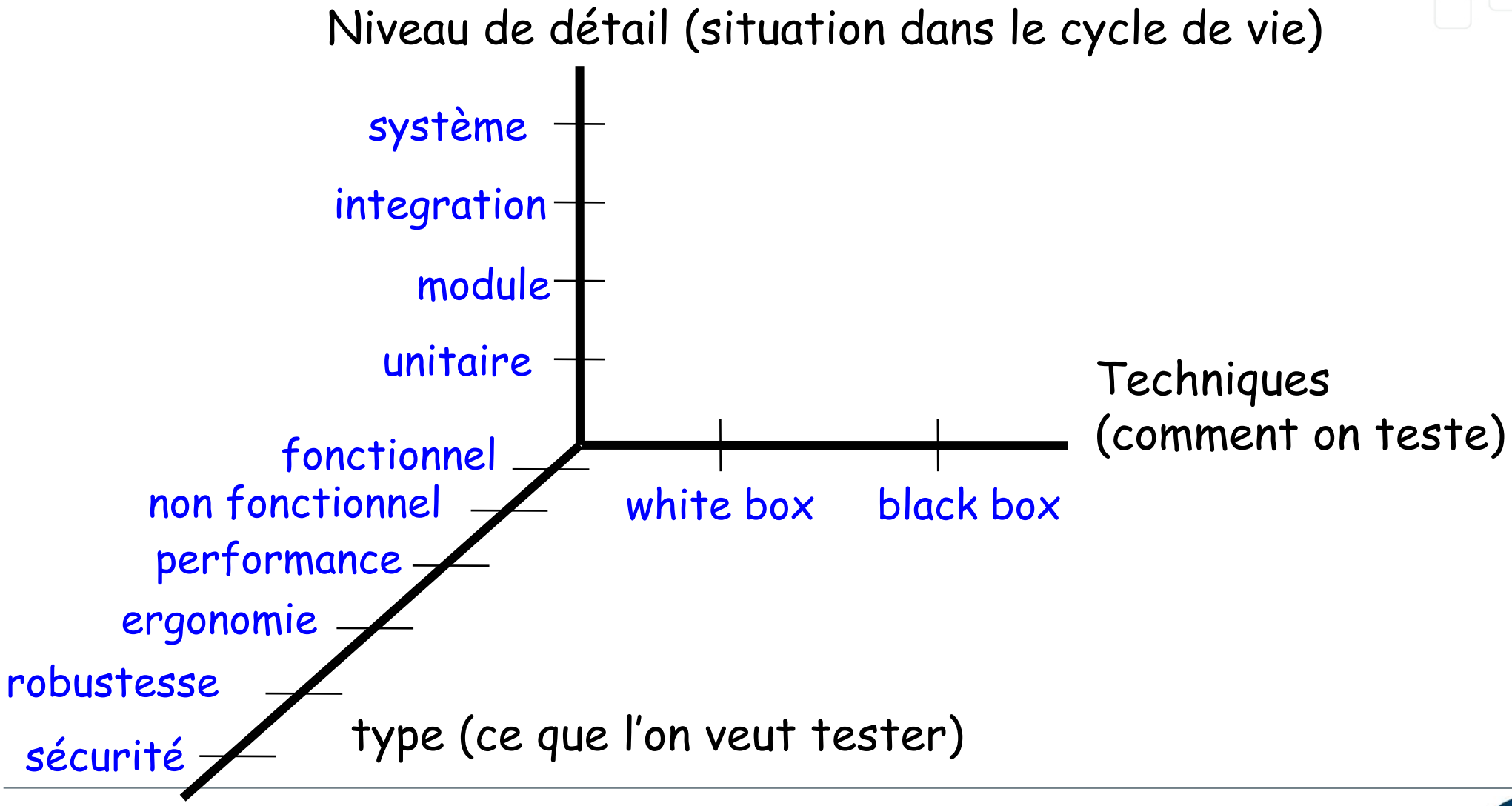
- **Test de performance**
  - Load testing (test avec montée en charge)
  - Stress testing (soumis à des demandes de ressources anormales)

- **Test de non-régression**
  Vérifie que les corrections ou évolutions dans le code n'ont pas créé d'anomalies nouvelles

- **Test de confirmation**
  Valide la correction d'un défaut

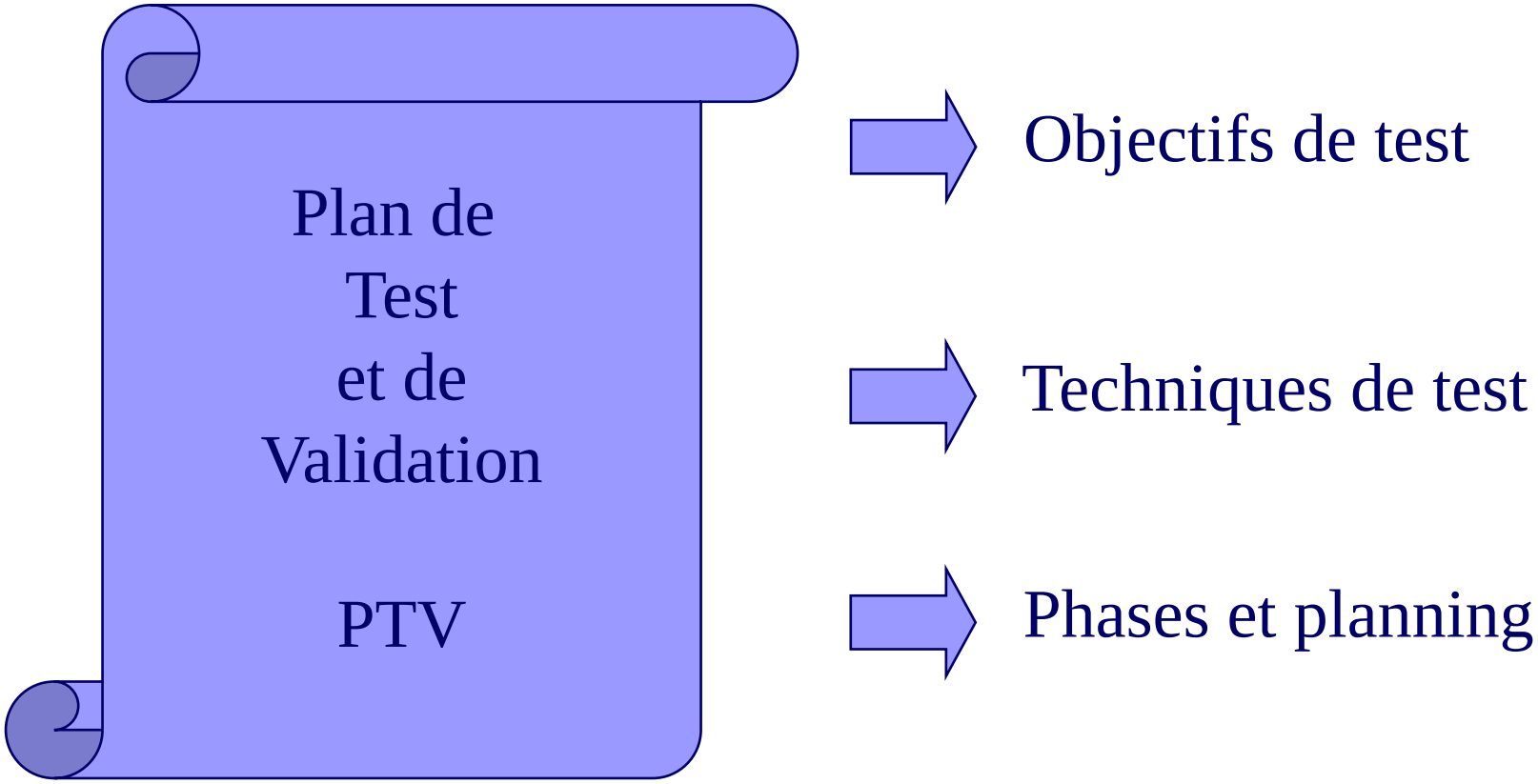
- …

Catégories de tests



Stratégie de test

- En début de projet, définition d'un Plan de Test et de Validation (PTV)



Acteur du test

- Deux situations :
  - Je teste un programme que j'ai écrit
  - Je teste un programme que quelqu'un d'autre a écrit

- Trois questions :
  - Comment choisir la technique de test ?
    => boite blanche ou boite noire ?
  - Comment obtenir le résultat attendu ?
    => problème de l'oracle du test
  - Comment savoir quand arrêter la phase de test ?
    => problème de l'arrêt

Test structurel

Fabrice AMBERT, Fabrice BOUQUET, Fabien PEUREUX,
Jean-Marie GAUTHIER, Alexandre VERNOTTE
prenom.nom@femto-st.fr

ENVOL 2014
La Londe-les-Maures
18-21 novembre 2014

Les bases du test structurel

- Le test structurel s'appuie sur l'analyse du code source de l'application pour établir les tests en fonction de critères de couverture

⇨ Basés sur le graphe de contrôle (toutes les instructions, toutes les branches, tous les chemins, …)

⇨ Basés sur la couverture du flot de données (toutes les définitions de variables, toutes les utilisations, …)

Graphe de contrôle

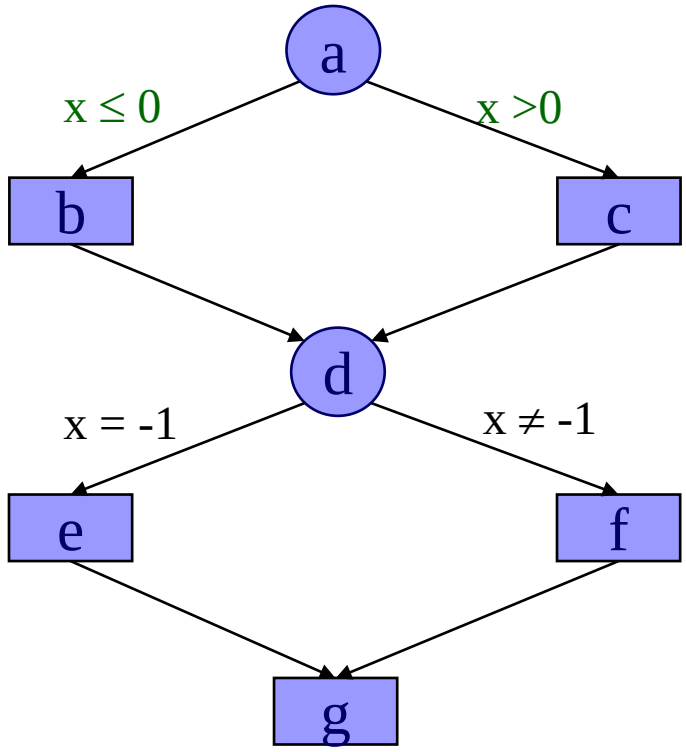
- Permet de représenter n'importe quel algorithme
- Les nœuds représentent des blocs d'instructions
- Les arcs représentent la possibilité de transfert de l'exécution d'un nœud à un autre
- Une seule entrée (nœud à partir duquel on peut visiter tous les autres) et une seule sortie

Graphe de contrôle – Exemple

Soit le programme P1 suivant :

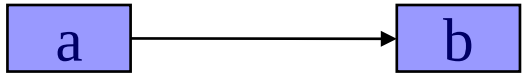
```
if x ≤ 0 then
    x := -x;
else
    x := 1 - x;

if x = -1 then
    x := 1;
else
    x := x+1;

writeln(x);
```

Et son graphe de contrôle G1 :

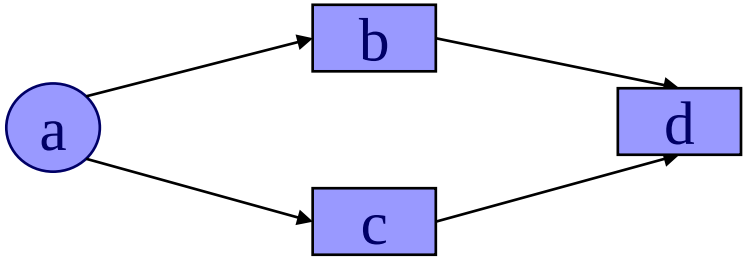


Expression des chemins de contrôle

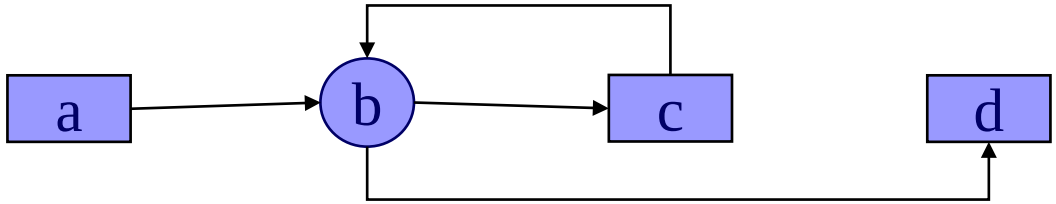
- On associe une opération d'addition ou de multiplication à toutes les structures primitives apparaissant dans le graphe de flot de contrôle
- Forme séquentielle : ab



- Instruction de décision : a (b+c) d

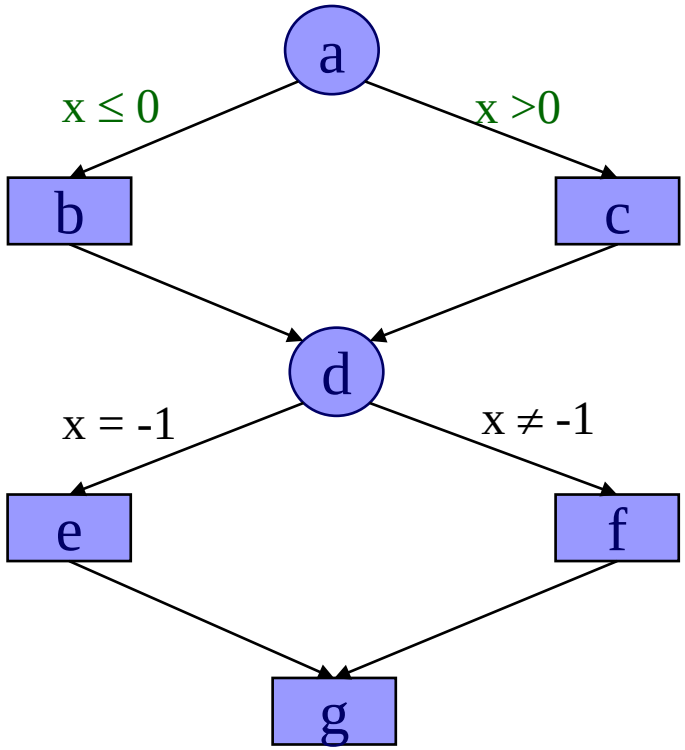


- Structures itératives : ab (cb)* d



Chemins dans le graphe de contrôle

- Le graphe G1 est un graphe de contrôle qui admet une entrée (le nœud a) et une sortie (le nœud g).
  - le chemin *[a, c, d, e, g]* est un chemin de contrôle,
  - le chemin *[b, d, f, g]* n 'est pas un chemin de contrôle.
- Le graphe G1 comprend 4 chemins de contrôle :
  - β1 = *[a, b, d, e, g]*
  - β2 = *[a, b, d, f, g]*
  - β3 = *[a, c, d, e, g]*
  - β4 = *[a, c, d, f, g]*

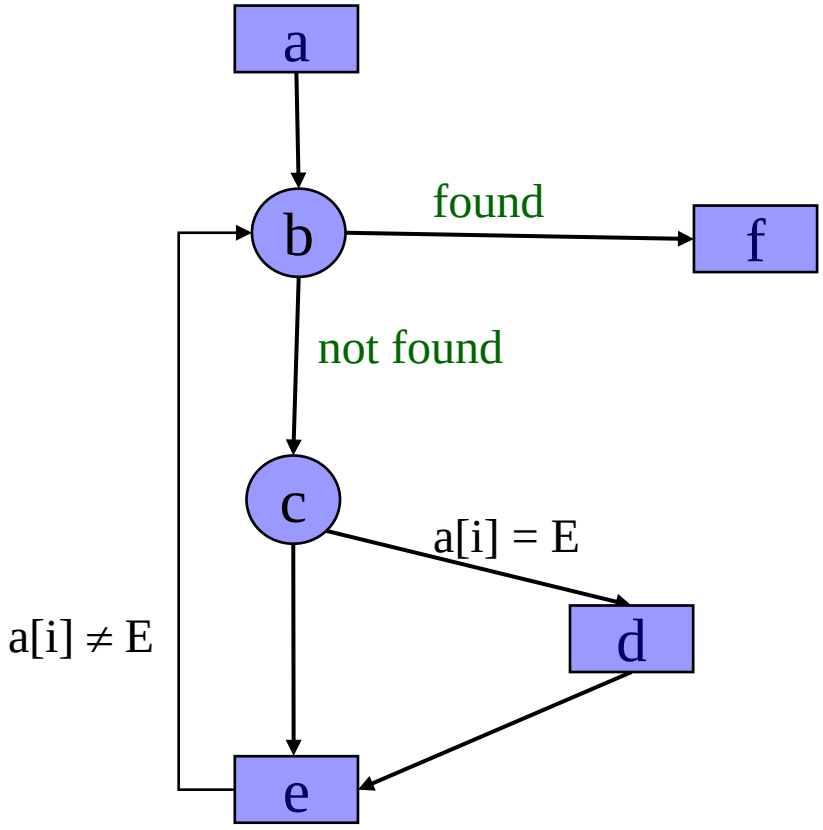


Graphe de contrôle – Exercice

Soit le programme P2 suivant :

```
i := 1;
found := false;
while (not found) do
   begin
      if (a[i] = E) then
         begin
            found := true;
            s := i;
         end;
      i := i + 1;
   end;
```

Et son graphe de contrôle G2 :



Production des jeux de tests à partir du graphe de contrôle

- On considère les chemins du graphe de contrôle (tous ou certains suivant le critère de couverture sélectionné)
- Des données d'entrée, permettant d'activer ces chemins, sont produites
- Le programme est exécuté dans ces configurations
- On recherche les anomalies de fonctionnement qui sont potentiellement détectables sur les chemins considérés

Critères de couverture basés sur le graphe de contrôle

- Couverture de *tous les nœuds*,
- Couverture de *tous les arcs*,
- Couverture des *chemins limites et intérieurs*
- Couverture de *tous les i-chemins*,
- Couverture de *tous les chemins indépendants*
- Couverture de *tous les chemins*

Couverture de tous les nœuds (ou couverture de toutes les instructions)

- Chaque nœud (chaque bloc d'instructions) est atteint par au moins l'un des chemins parmi les chemins qui constituent le jeu de tests

- Lorsqu'un jeu de test permet de couvrir tous les nœuds du graphe, on dit qu'il satisfait TER=1 ou TER1 (Test Effectiveness Ratio 1)

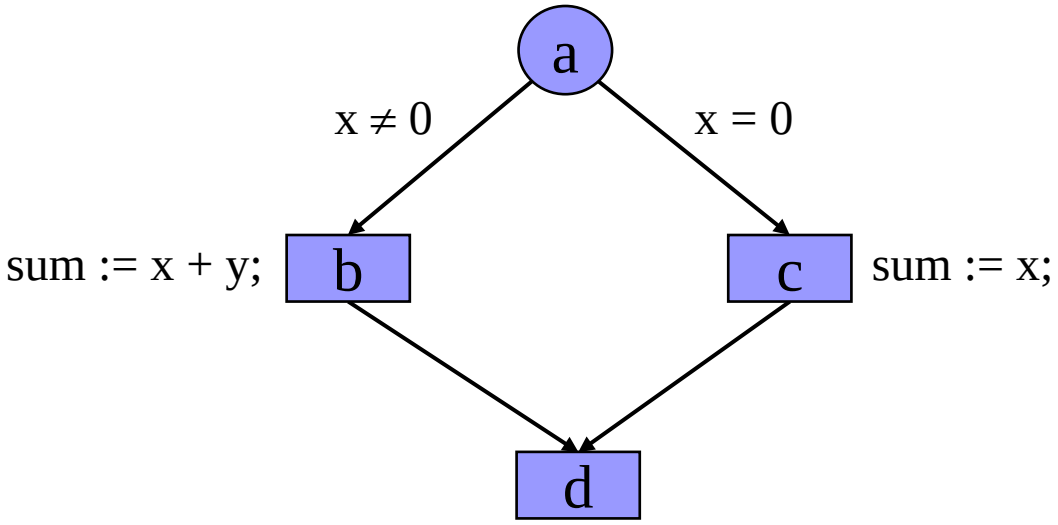
  TER = 1 <=> le critère *tous les nœuds* est satisfait
  <=> tous les nœuds du graphe de contrôle ont été couverts
  <=> toutes les instructions ont été exécutées

Couverture de tous les nœuds Exemple

- Taux de couverture : $\frac{\text{nb de nœuds couverts}}{\text{nb total de nœuds}}$

- Soit le programme suivant (somme avec erreur) :

```
function sum (x,y : integer) : integer;
    begin
        if (x = 0) then
            sum := x ;
                else
            sum := x + y ;
              end ;
```

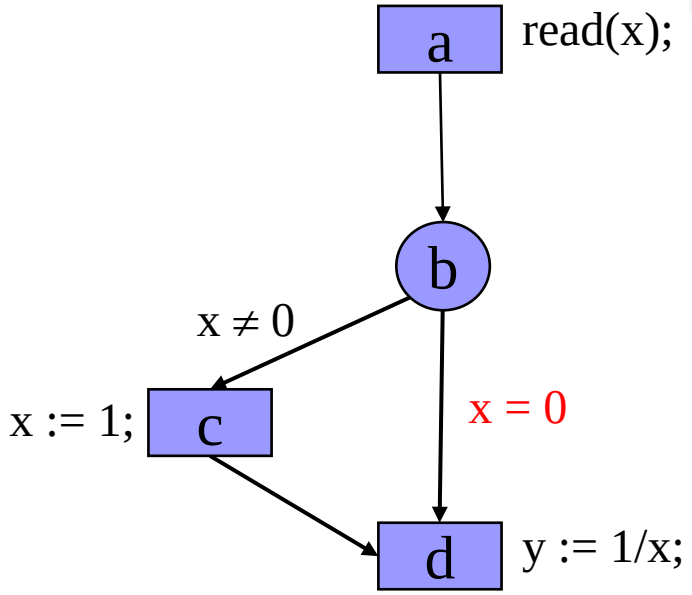


⇒ L 'erreur est détectée par l'exécution du chemin [acd]

Couverture de tous les nœuds
Limites de ce critère

Soit le programme suivant (avec erreur) :

read (x) ;
if (x ≠ 0) **then**
    x := 1;
y := 1/x;



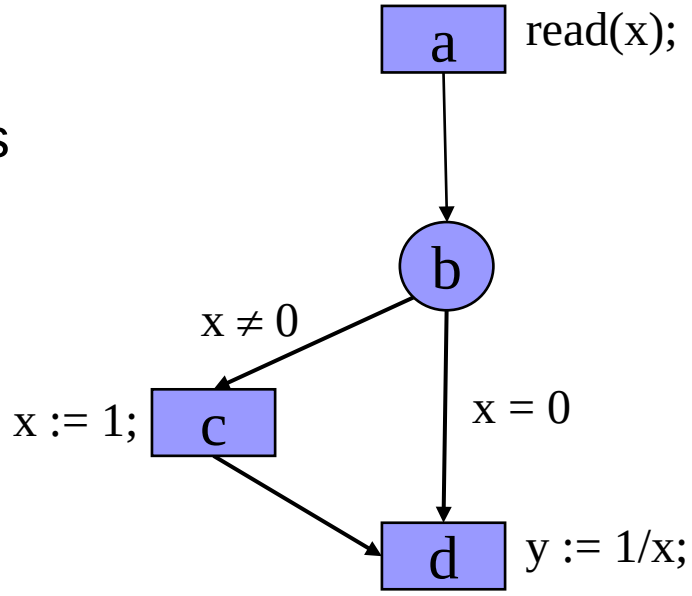
⇒ Le critère *tous-les-nœuds* est satisfait par le chemin [abcd] sans que l'erreur ne soit détectée.
L'unique donnée de test {x = 2} permet de couvrir tous les nœuds du graphe sans faire apparaître l'anomalie.

Couverture de tous les arcs (ou couverture de tous les enchaînements)

- Taux de couverture : $\frac{\text{nb d'arcs couverts}}{\text{nb total d'arcs}}$

- Impose la couverture de tous les chemins constitués uniquement d'un arc

Sur l'exemple précédent, le jeu de tests doit permettre de couvrir au moins une fois les chemins *ab*, *bc*, *cd* et *bd* pour satisfaire ce critère.



Couverture de tous les arcs (ou couverture de tous les enchaînements)

- La couverture de tous-les-arcs équivaut à la couverture de toutes les valeurs de vérité pour chaque nœud de décision (leur valeur de vérité à été au moins une fois vraie et une fois fausse)

- Lorsqu'un jeu de tests permet de couvrir tous les arcs du graphe, on dit qu'il satisfait TER=2 ou TER2 (Test Effectiveness Ratio 2)

  TER = 2 <=> le critère *tous-les-arcs* est satisfait

  <=> tous les arcs du graphe de contrôle ont été couverts

  <=> toutes les décisions ont été exécutées

- Un jeu de test qui satisfait le critère TER2, satisfait également le critère TER1.

Limites des critères tous les arcs

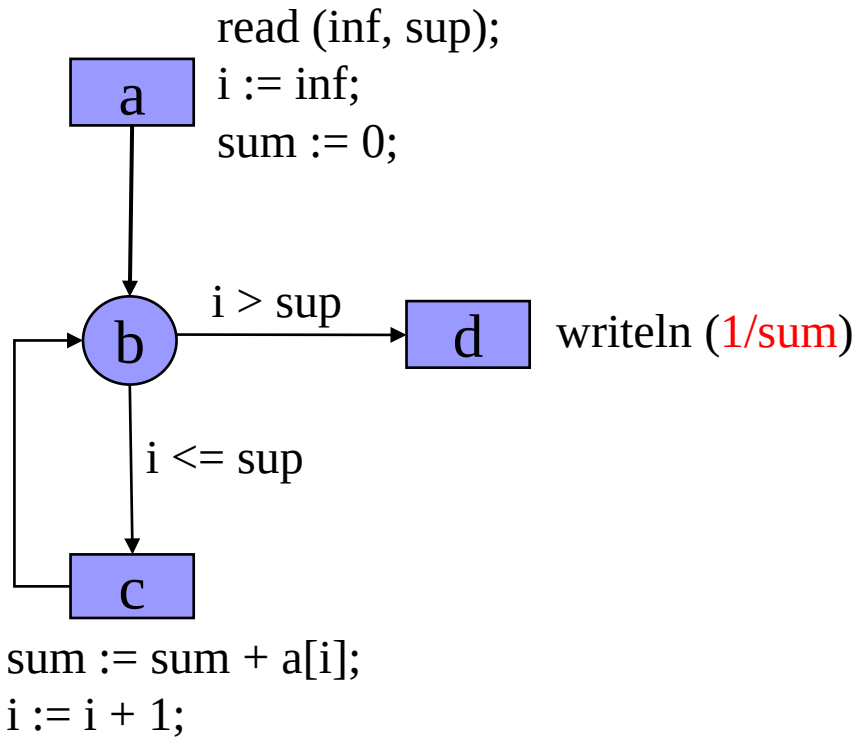
- Pas de détection d'erreurs **en cas de non-exécution d'une boucle**

⇒ Le critère *tous-les-arcs* est satisfait par le chemin [abcbd]

⇒ La donnée de test suivante couvre le critère *tous-les-arcs* :

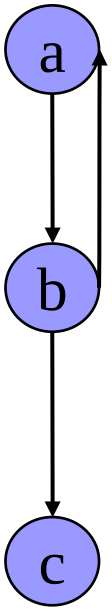
DT1= {a[1]=50, a[2]=60, a[3]=80, inf=1,sup=3}

⇒ Problème non détecté par le critère *tous-les-arcs* : si inf > sup, erreur sur 1/sum

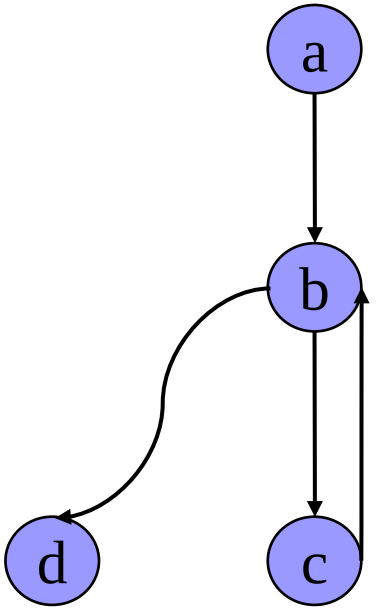


Couverture des boucles

- Chemins limites : traversent la boucle, mais ne l'itèrent pas
- Chemins intérieurs : itèrent la boucle une seule fois



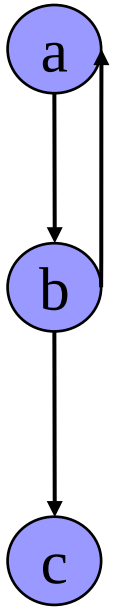
Chemin limite : [abc]
chemin intérieur : [ababc]



chemin limite : [abd]
chemin intérieur : [abcbd]

Couverture de tous les i-chemins

- Couverture de tous les chemins possibles passant de 0 à i fois dans chaque boucle du graphe de contrôle
- La couverture de *tous les i-chemins* (pour i > 0) garantit les critères TER1, TER2 et le critère de couverture des chemins limites et intérieurs



Le jeu de tests constitué de données de test permettant de couvrir les chemins [ABC], [ABABC] et [ABABABC] satisfait le critère *tous les 2-chemins.*

Couverture de tous les chemins indépendants

- Le critère **tous les chemins indépendants vise à parcourir tous les arcs dans chaque** configuration possible (et non pas au moins une fois comme dans le critère tous-les arcs)

- Le nombre de chemins indépendants d'un graphe G est donné par le nombre de McCabe, également appelé nombre de cyclomatique, noté V(G)
  V(G) = Nb d'arcs - Nb de nœuds + 2

Couverture de tous les chemins indépendants – procédure

1. Evaluer V(G)
2. Produire une donnée de test au hasard couvrant le maximum de nœuds de décisions du graphe
3. Produire une donnée de test qui modifie le flot de la première instruction de décision du graphe
4. Vérifier l'indépendance du chemin calculé par rapport aux autres chemins déjà calculés (2 chemins sont indépendants s'il existe un arc qui est couvert par l'un et pas par l'autre)
5. Recommencer les étapes 3 et 4 jusqu'à la couverture de toutes les décisions
6. S'il n'y a plus de décisions à étudier sur le chemin initial et que le nombre de chemins trouvés est inférieur à V(G), considérer les chemins secondaires déjà trouvés

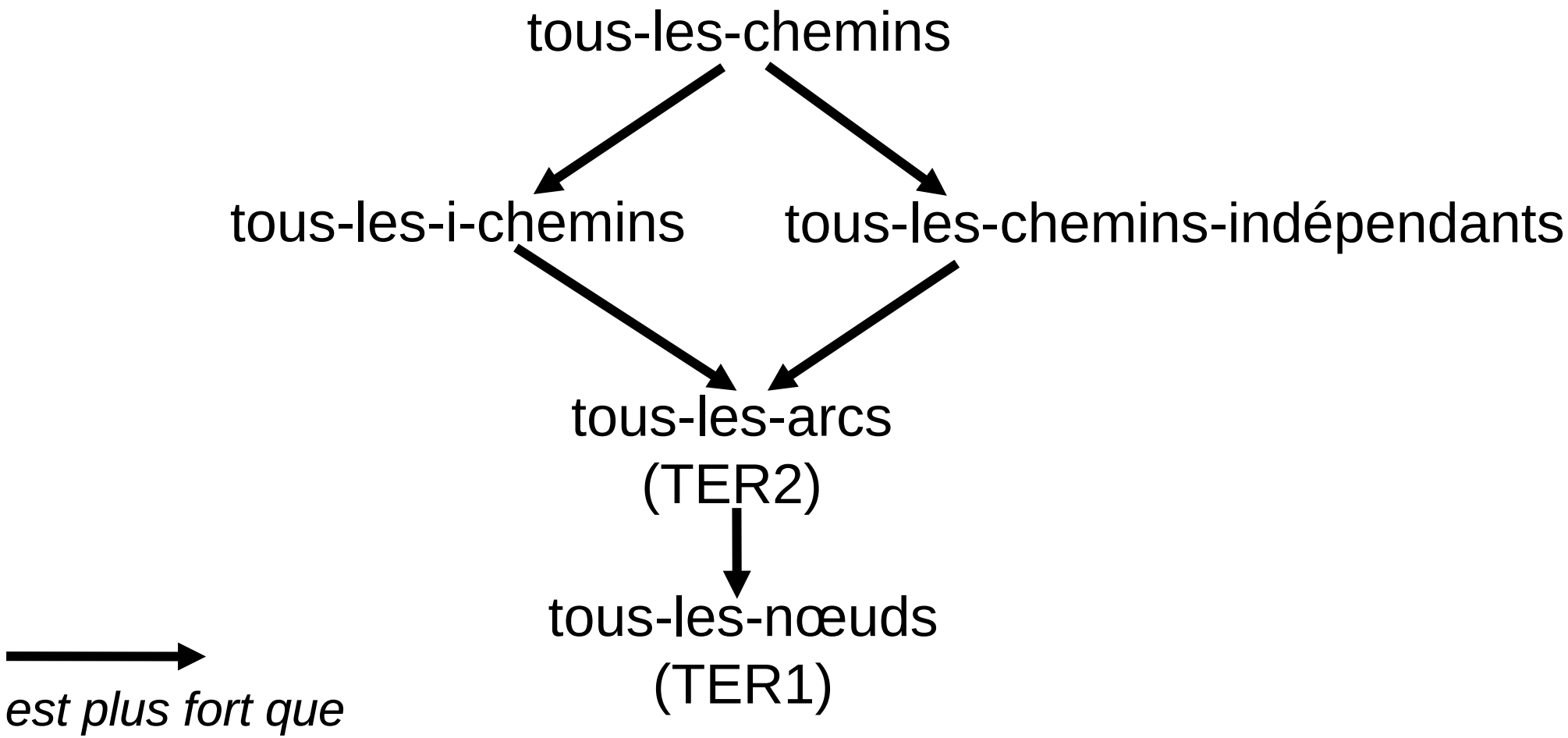
Couverture de tous les chemins

- En posant *i=n* (avec *n* le nombre maximal d'itérations possibles) pour le critère *tous les i-chemins*, cela revient à exécuter tous les chemins possibles du graphe de contrôle

  => Critère *tous-les-chemins*

- En pratique, ce critère est rarement envisageable sur des applications même modestes (explosion du nombre de données de test nécessaires)

Hiérarchie des critères basés sur le graphe de contrôle



Le flot de données

- Le flot de données est représenté en annotant le graphe de contrôle par certaines informations sur les manipulations de variables par le programme :
  - Définition de variables : la valeur de la variable est modifiée

  Exemple : membre gauche d'une affectation, paramètre d'une instruction de lecture…
  - Utilisation de variables : accès à la valeur de la variable

  Exemple : membre droit d'une affectation, paramètre d'une instruction d'écriture, indice de tableau, utilisée dans une condition dans une instruction conditionnelle, dans une boucle…

- Si une variable utilisée l'est dans le prédicat d'une instruction de décision (if, while, etc), il s'agit d'une *p-utilisation*
- Dans les autres cas (par exemple dans un calcul), il s'agit d 'une *c-utilisation.*

Définition et utilisation de variables

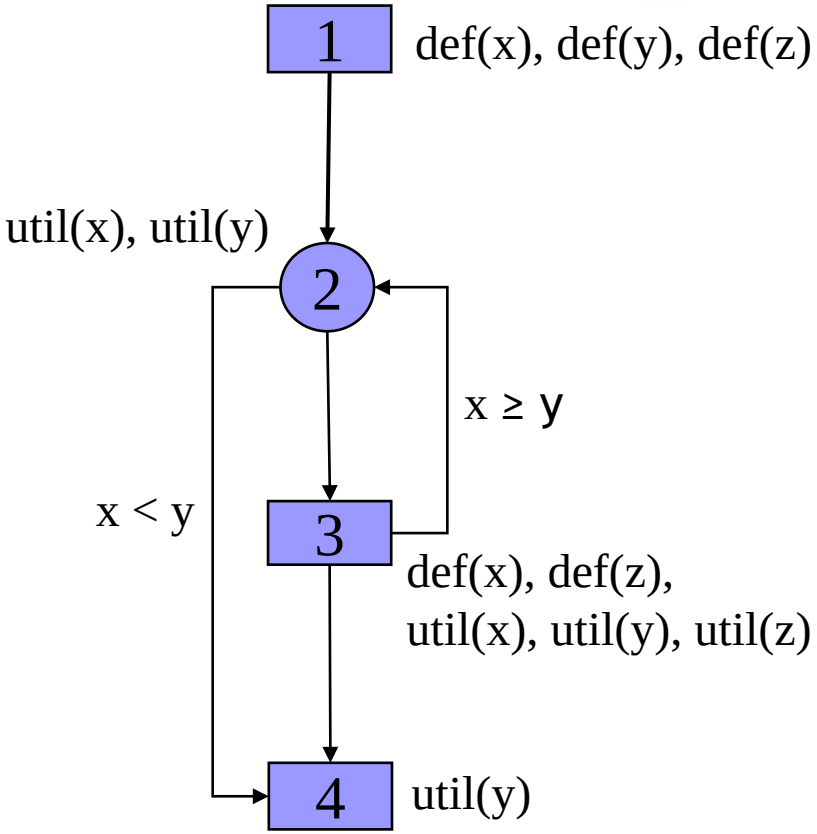
```
open (fichier1) ;
read (x,fichier1) ;
read (y,fichier1) ;
z := 0 ;
---------------------------
while x ≥ y do
---------------------------
    begin
        x := x-y ;
        z := z+1 ;
    end
---------------------------
open (fichier2) ;
print (y,fichier2) ;
close (fichier1) ;
close (fichier2) ;
```

Bloc 1
Définitions : x, y, z
Utilisations : aucune

Bloc 2
Définitions : aucune
Utilisations : x, y

Bloc 3
Définitions : x, z
Utilisations : x, y, z

Bloc 4
Définitions : aucune
Utilisations : y

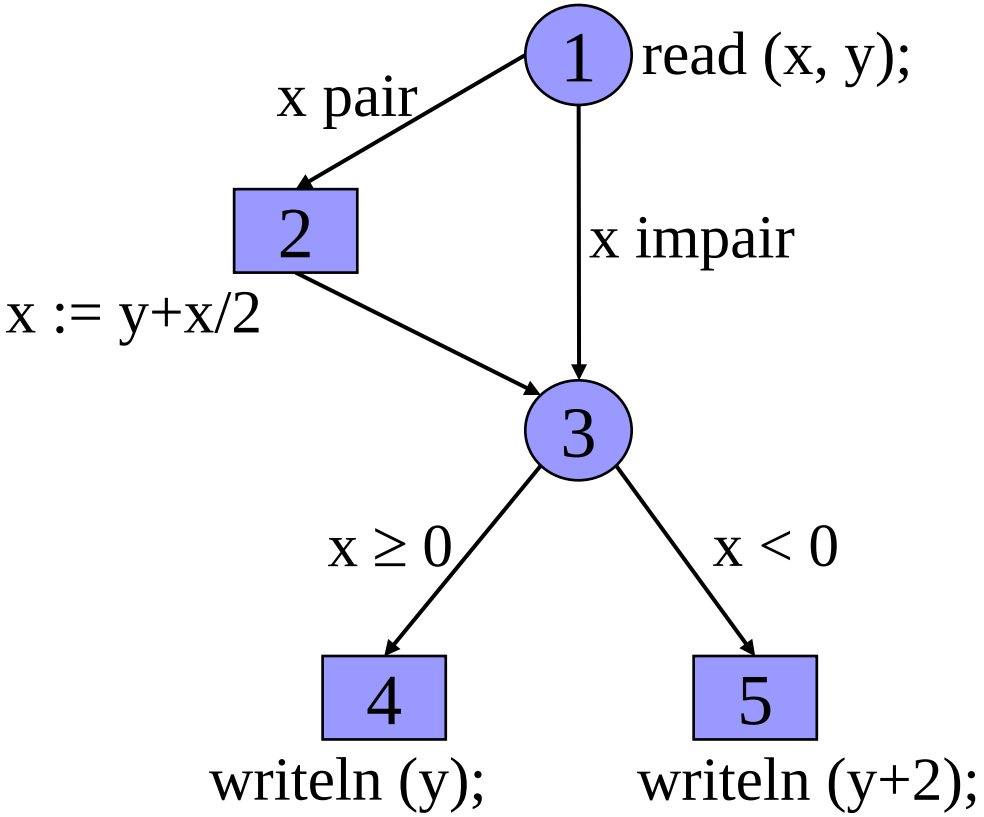


Critères de couverture sur le flot de données

- Toutes les définitions
- Toutes les utilisations
- Tous les DU-chemins

Couverture de toutes les définitions

- Pour chaque définition, il existe au moins un chemin qui le couvre dans un test

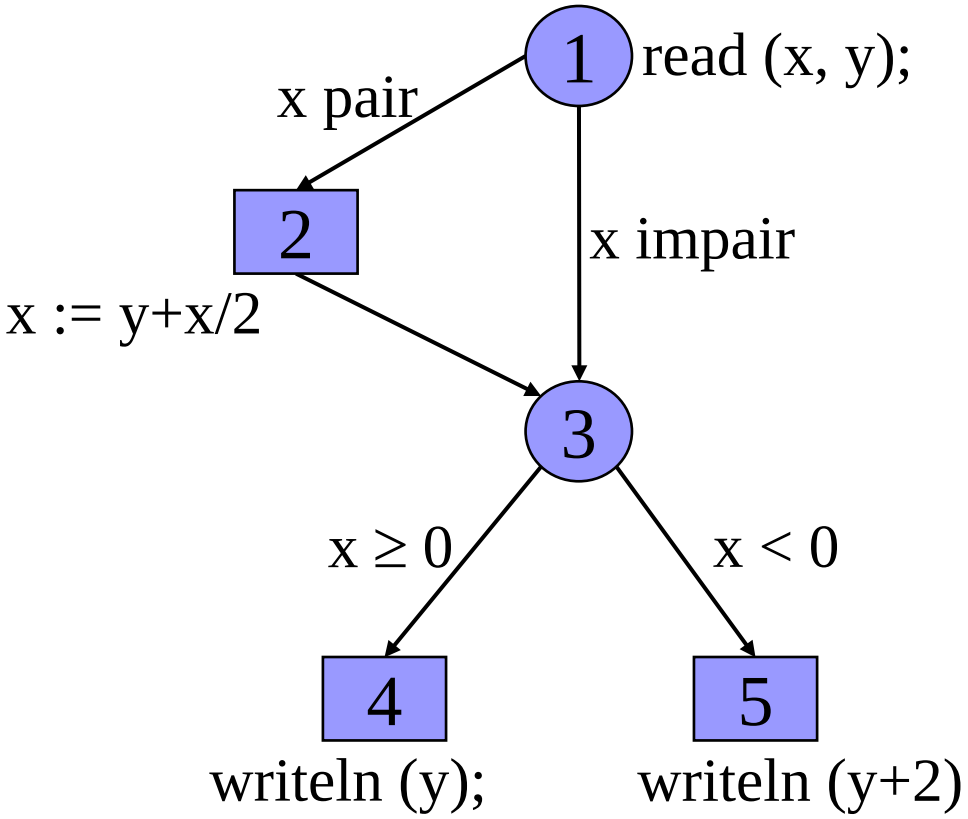


Couverture du critère toutes les définitions :

- [1,2,3,5]

Couverture de toutes les utilisations

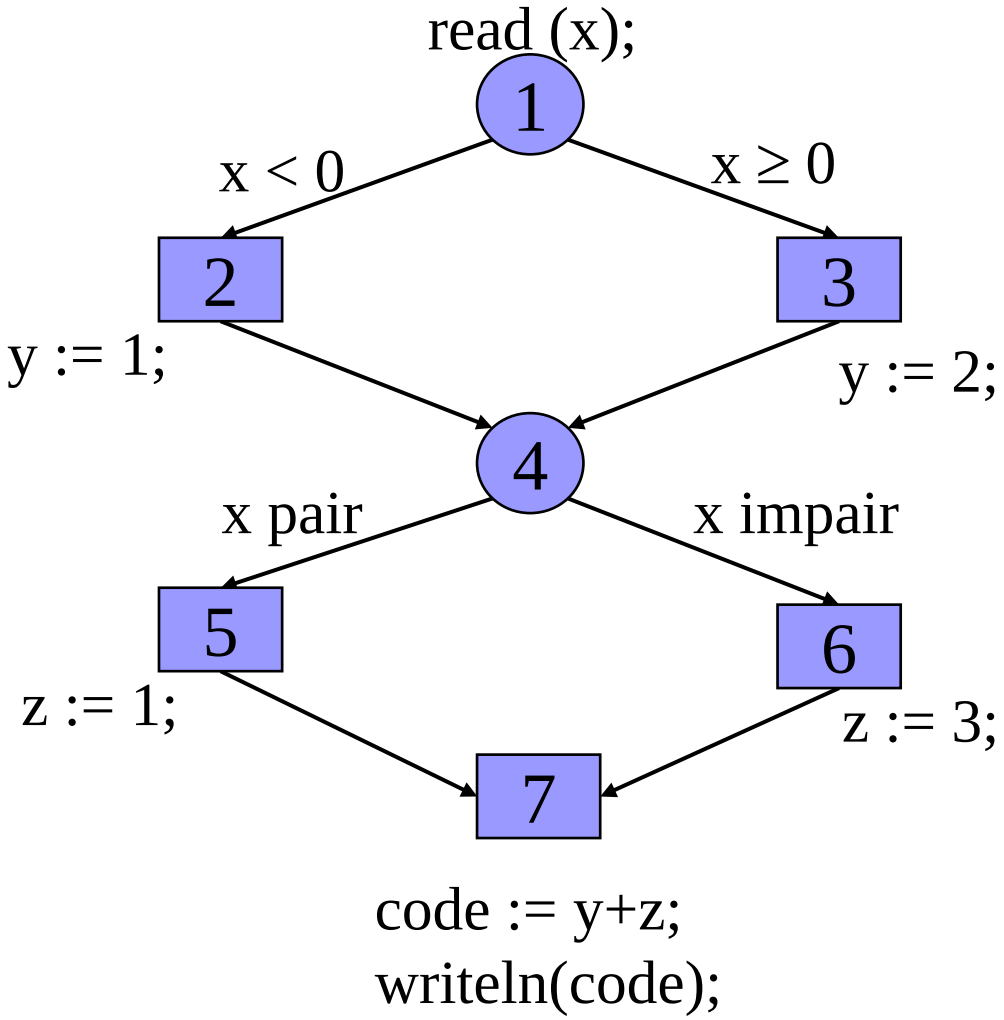
- Pour chaque définition et pour chaque utilisation accessible à partir de cette définition, il existe un test qui exerce l'utilisation (sans redéfinition)



Couverture du critère toutes les utilisations :

- [1,3,4]
- [1,2,3,4]
- [1,3,5]
- [1,2,3,5]

Limite du critère toutes les utilisations



Couverture du critère toutes les utilisations :

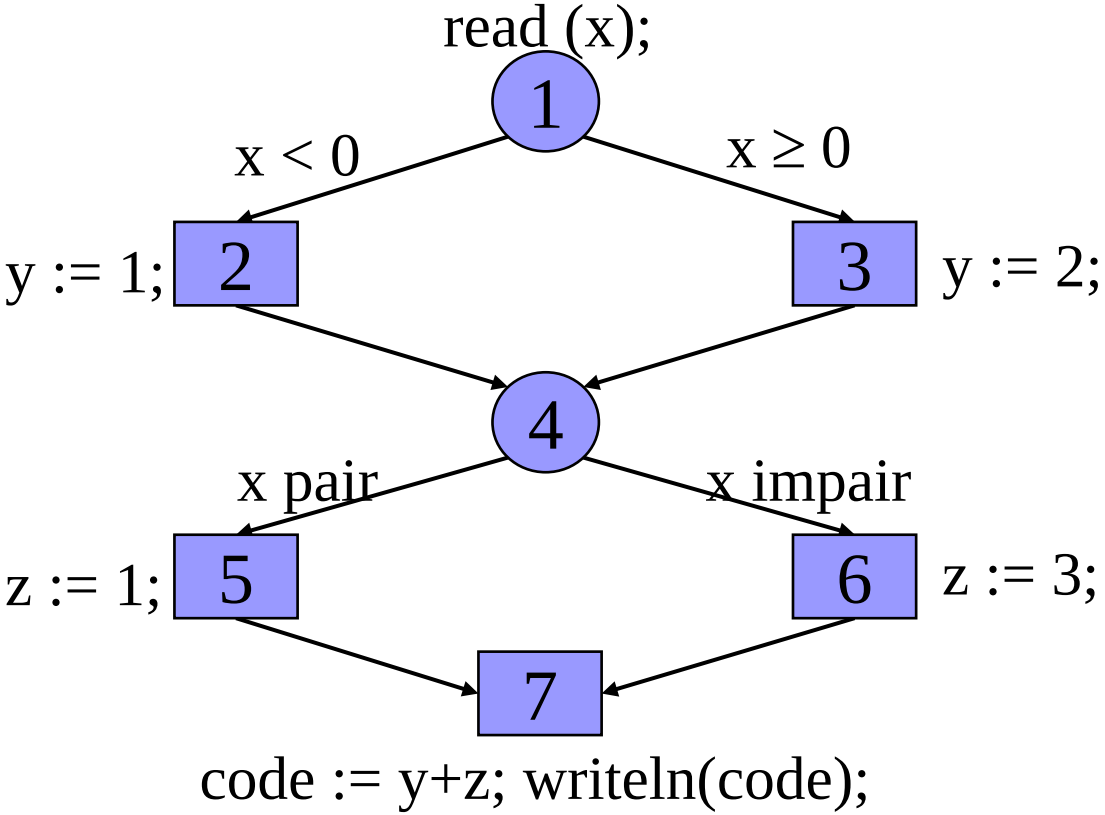
- [1,2,4,5,7]
- [1,3,4,6,7]

Ces deux tests ne couvrent pas tous les chemins d'utilisation :
(7) peut être utilisatrice de (2) pour la variable y et de (6) pour la variable z

=> critère *tous les DU-chemins*

Couverture de tous les DU-chemins

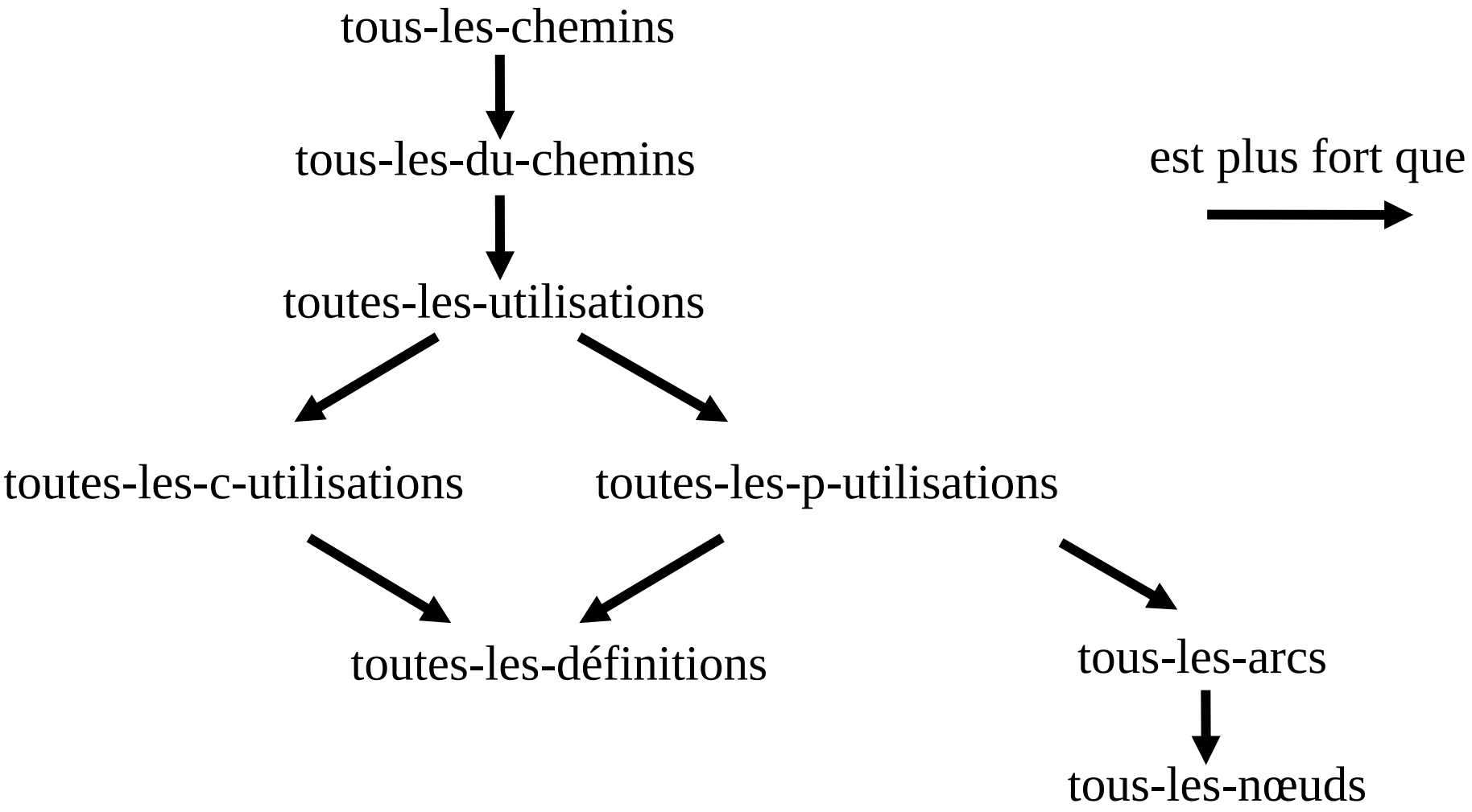
- Ce critère rajoute au critère *toutes les utilisations* le fait qu'on doit couvrir tous les chemins possibles entre la définition et la référence, en se limitant aux chemins sans cycle.



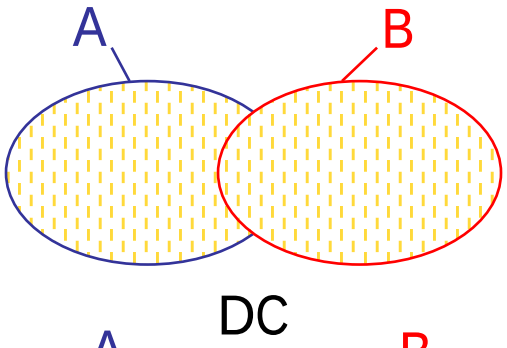
Couverture du critère tous les DU-chemins :

- [1,2,4,5,7]
- [1,3,4,6,7]
- [1,2,4,6,7]
- [1,3,4,5,7]

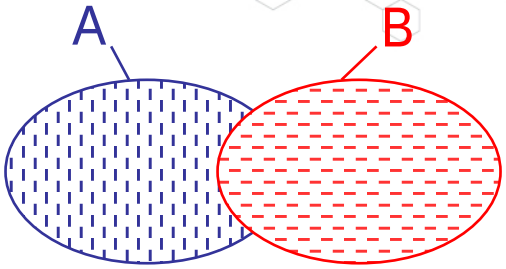
Hiérarchie des critères basés sur le flot de données



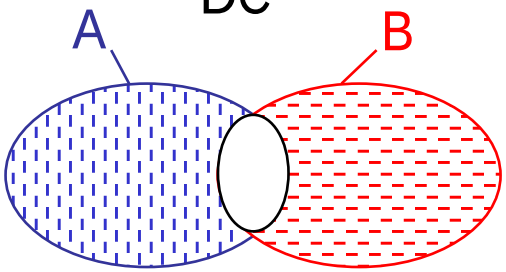
Couverture des Conditions / Decisions



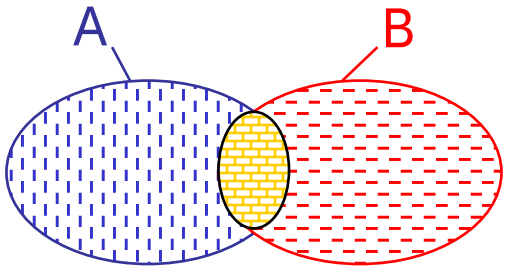
DC



C/DC



FPC



MCC

Décision ($A \vee B$) :

- Couverture des Décisions → $A \vee B$
- Couverture des Decisions/Conditions → A , B
- Full Predicate Coverage → $A \wedge \neg B$, $\neg A \wedge B$
- Multiple Condition Coverage → $A \wedge B$, $A \wedge \neg B$, $\neg A \wedge B$

Synthèse

• La production de test s'appuie (généralement) sur une analyse du programme (test structurel) ou de sa spécification (test fonctionnel)

• Différentes stratégies permettent de sélectionner des données de test pertinentes.

• Ces stratégies ne sont que des heuristiques !
Elles ne fournissent aucune garantie de sélectionner la valeur qui révélera les erreurs du programme.

Programme

- Mercredi 19/11/14 : test structurel
  - Exercice de couverture
  - Java : Junit, Mockito, Jacoco
  - C++ : CPPUnit, GoogleMock, Gcov
- Jeudi 20/11/14 : test fonctionnel
  - ALL : Squash TM, Selenium
  - Java : Concordion
  - C++ : Cucumber

Merci pour votre attention…