UE Génie logiciel 2
Vendredi 2 décembre 2021

Test du logiciel

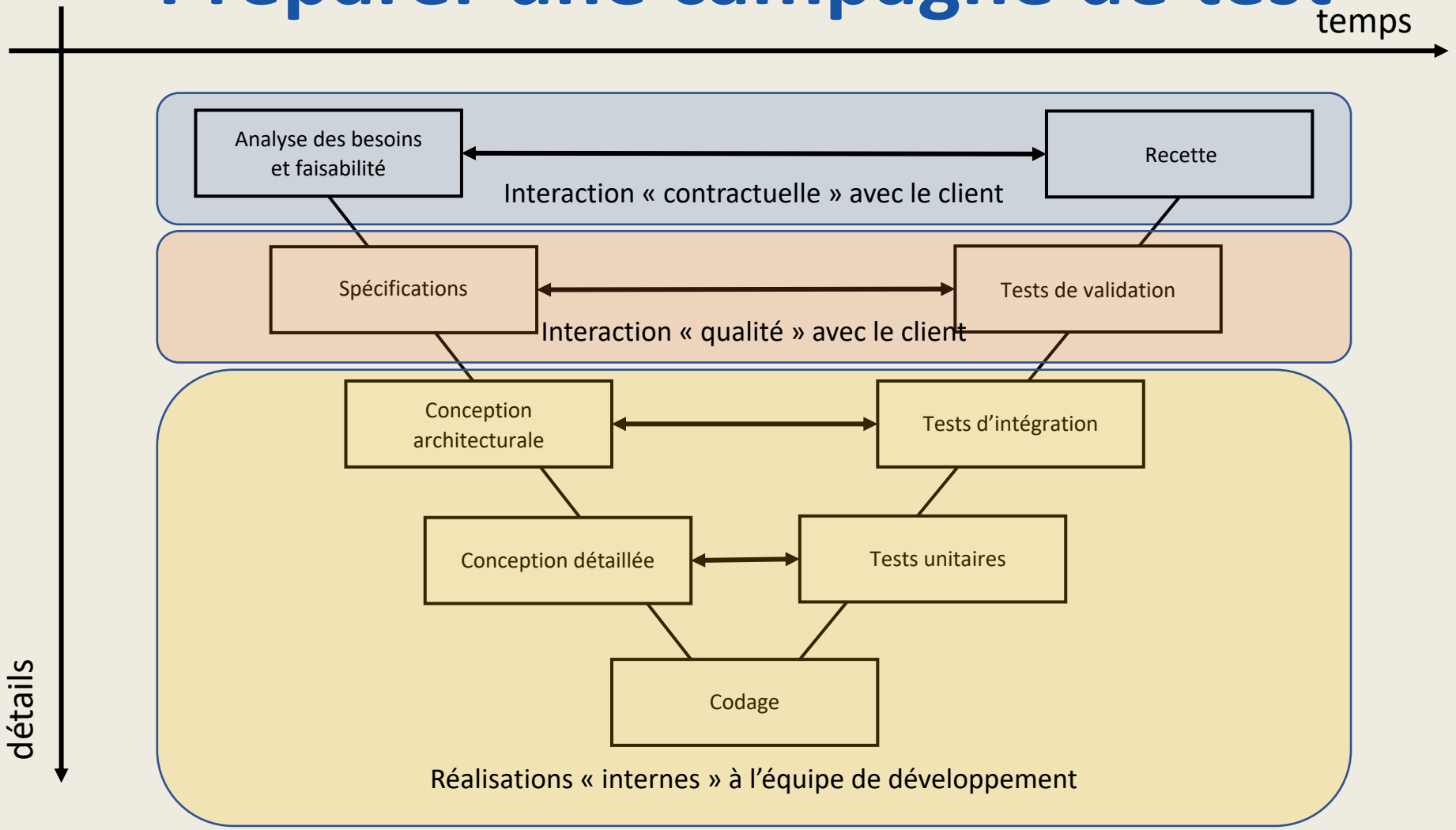
Damien MONDOU, Enseignant chercheur, La Rochelle Université
damien.mondou@univ-lr.fr

La Rochelle Université

La Rochelle Université

PARTIE I : INTRODUCTION

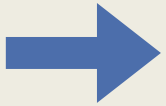
Préparer une campagne de test



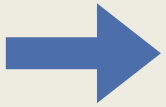
Chaque étape descendante du cycle en V doit donner lieu à la production de jeux de tests qui seront mis en œuvre et exécutés dans la phase montante pour validation du logiciel.

Le test

Définitions :



« 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 résultat attendus et les résultats obtenus » IEEE - Standart 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, 1979

La Rochelle Université

Modalités du test

Test statique : sans exécution du code

- Relecture / revue de code
- Analyse automatique (vérification de propriétés, règles de codage, etc.)
- Analyse des chemins

Test dynamique : avec exécution du code

- Exécution du programme avec des valeurs en entrée et observation de la sortie (ex : Junit)
- Test partitionnel
- Test aux limites

La Rochelle Université

Principes du test

→ Expérimenter toutes les exécutions possibles

- Comparer les résultats obtenus avec les résultats attendus
- Difficulté de calculer précisément les résultats attendus

→ Tester tous les cas : difficilement réalisable (voir impossible)

- Combinatoire trop importante
- Solution : accepter la non exhaustivité, sélectionner les jeux de test

→ Alternative

- Choix d'une couverture raisonnable
- Test structuré et méthodique

La Rochelle Université

Principes du test

- Les tests doivent vérifier le domaine valide …
- …, le domaine invalide …
  - Entrées au format ou au type incorrect
  - Evénement inattendu
  - Non respect des pré-conditions
- … et également les conditions limites (valides et invalides)

La Rochelle Université

Ce qu'il faut tester

- Fonctionnalités
- Sécurité / intégrité
- Utilisabilité
- Cohérence
- Maintenabilité
- Efficacité
- Robustesse
- Sûreté de fonctionnement
- Performances

La Rochelle Université

La Rochelle Université

PARTIE II : TESTS BOITE BLANCHE ET BOITE NOIRE

Boite blanche / boite noire

- **Tests en boite noire** : Point de vue client : vérification de la fonctionnalité du logiciel (entrées et sorties)
- **Tests en boite blanche** : Le code est connu et analysé
- **Tests en boite grise** : mélange des 2 : Vérification de la fonctionnalité et vérification du processus permettant d'y arriver

La Rochelle Université

Tests en boite noire = Tests fonctionnels

- Validation fonctionnelle d'une application d'un point de vue **externe**
- Vérification de la prise en compte des **exigences fonctionnelles** et/ou du respect des **fonctionnalités métiers**.
- Permet de détecter une erreur fonctionnelle dans le système

Exemple :

- Fonction prévue mais non implémentée
- Calcul erroné : la sortie ou l'événement est produit mais les valeurs sont fausses
- Refus d'une entrée appartenant au domaine valide

La Rochelle Université

Tests en boite noire = Tests fonctionnels

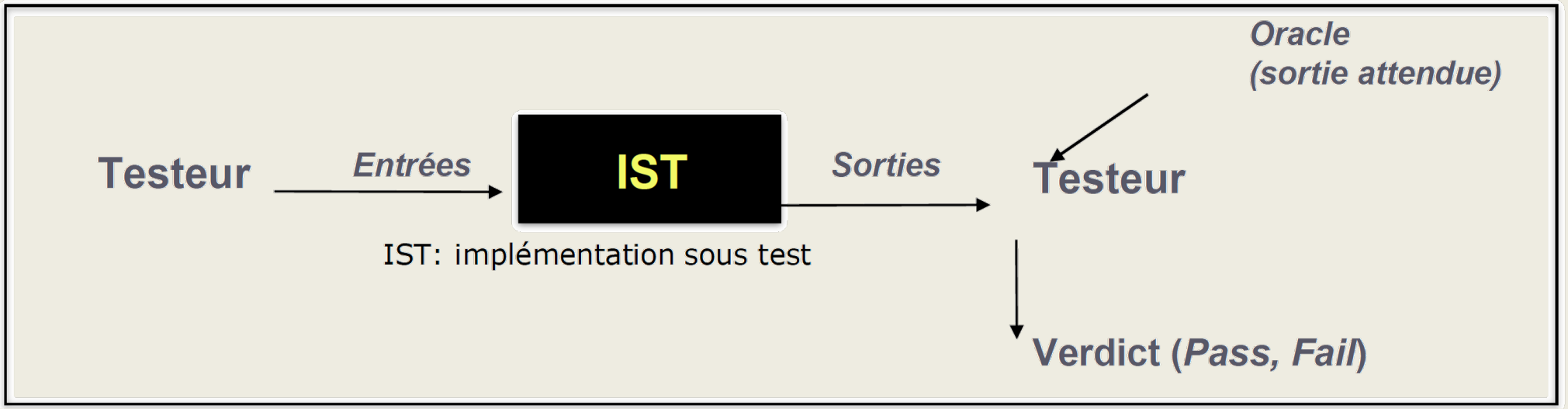
- Le cas de test est conçu à partir des spécifications du logiciel (par exemple les cas d'utilisation définis par UML)

- L'écriture des scénario de test est facilitée car issu des spécifications. Idem pour les sorties attendues.

- Il peut être difficile de concrétiser les **données de test** sans passer par une analyse approfondie des documents de conception (ambiguïté des spécifications)

La Rochelle Université

Tests en boite noire : l'exécution



Le testeur émet des entrées vers l'IST (considéré comme une boite noire) et analyse les sorties produites en les comparant avec l'oracle pour établir le verdict de test.

Analyse partitionnelle, tests aux limites

Analyse des E/S

Analyse combinatoire

Tests de la qualité du système au sens large (performances, etc.)

La Rochelle Université

Tests en boite noire : ce qu'il ne peuvent pas faire

Garantir l'absence d'erreurs dans le système: le test ne «prouve» pas la qualité d'un logiciel mais met en évidence des erreurs :

- une suite exhaustive de tests n'existe pas (ex : un calcul pour des valeurs entières non bornées)
- L'exhaustivité de l'ensemble des exigences dans le cahier des charges est une hypothèse
- Impossible de tester l'absence de deadlocks

Si l'ensemble de la revue de test ne produit **aucun résultat négatif**…

- On peut conclure que
  - **Les propriétés désignées par les tests sont vraies pour ce système**
- On ne peut pas conclure que
  - le système ne contient aucune erreur (test non exhaustif.)

La Rochelle Université

Tests en boite blanche = Tests structurels

→ Le code est connu et analysé

→ Examiner le fonctionnement d'une application et sa structure interne, ses processus, plutôt que ses fonctionnalités.

Avantages :

- On peut anticiper en effectuant ces tests tout au long du développement.
- On peut optimiser son code, améliorer les performances
- On peut tester intégralement le code : détection des bugs, vulnérabilités cachées

La Rochelle Université

Tests en boite blanche = Tests structurels

→ Le code est connu et analysé

→ Examiner le fonctionnement d'une application et sa structure interne, ses processus, plutôt que ses fonctionnalités.

Inconvénients :

- Avoir des connaissances en programmation (≠ tests boite noire)
- Longue durée des tests (lié à la longueur du code)
- Cadrage difficile : qu'est ce qu'il faut tester ? Qu'est ce qu'on peut laisser de côté ?

Comment faire ? :

Utilisation des parcours de graphes pour

- Tester le passage par les chemins
- Tester l'absence de code « mort » ou redondant

La Rochelle Université

Types de test

- **Test de robustesse** : Test dans des conditions de fonctionnement erronées (pas d'accès au réseau, à une base de données, etc.)
- **Test de performance** : Valider la capacité qu'a l'architecture à supporter les charges et à faire fonctionner l'application (efficacité, temps de réponse)
- **Test de charge ou aux limites** : Simuler une activité égale ou supérieur à celle en conditions normales
- **Test de crash**: Tester les réactions en cas de crash disque, coupure électrique, etc.
- **Test de non régression** : Vérifier que les modifications n'ont pas altérées le fonctionnement de l'application

La Rochelle Université

Tests unitaires

Objectif : Tester des portions du code (fonctions, classes)

Nom de l'objet : MonObjet
Nom de la méthode : maMéthode **Paramètres de la méthode** : String
Objet retourné par la méthode : String **Exceptions possibles** : MonException

Scenario 1 :
- **Paramètres** : null
- **Résultat attendu** : une exception de type MonException est lancé avec le message d'erreur suivant : "Paramètre incorrect"
- **Résultat du test** : OK
- **Date du test** : 03/12/2021
- **Testeur** : M. Untel

La Rochelle Université

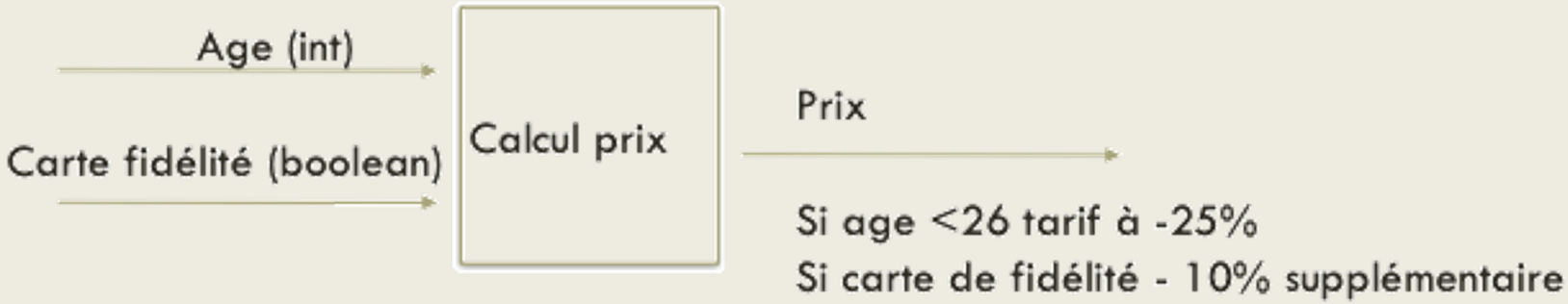
La Rochelle Université

PARTIE III : LES MÉTHODES DE TEST

Test combinatoire

Pour garantir la qualité du programme, il faut exécuter toutes les séquences de test => IMPOSSIBLE dans le cas de test sur les valeurs

Solution : supposer que la correction du programme pour certaines séquences de test entraine sa correction pour d'autres séquences de test "équivalentes"



La Rochelle Université

Test combinatoire

Définition: Un test combinatoire est un test exhaustif sur une sous-partie délimitée des combinaisons possibles des valeurs d'entrée

Exemple : calcul du prix du billet de train (pour un billet à 100€ en plein tarif)

- Combinatoire : n variables booléennes en entrées
- Principe : construire la table de vérité (2^n données de test)

	Age < 26	Carte	Données de test (DT)	Résultat attendu	Résultat observé	Résultat du test
DT1	0	0	28, false	100	100	PASS
DT2	0	1	30, true	90	90	PASS
DT3	1	0	24, false	75	75	FAIL

La Rochelle Université

Le nombre de tests à effectuer peut être très important en fonction du nombre d'entrées : Réduction par pair-wise

Test combinatoire : Réduire le nombre de test par Pair-Wise

Pair-Wise : Sélection de tests pour couvrir toutes les paires de valeurs

Hypothèse : Majorité des fautes détectées par des combinaisons de deux paires de valeurs

Procédure :

- Identifier les paires de variables
- Identifier le nombre de paires de valeurs = nb de paires de variables x 4
- Identifier les paires de valeurs qui permettent de tout couvrir

La Rochelle Université

Test combinatoire : Réduire le nombre de test par Pair-Wise - Exemple

- Exemple pour 3 variables en entrées (M, L, C)
- Paires de variables : (M, C), (M, L), (C, L)
- Paires de valeurs possibles : (M=0, C=0), (M=0, C=1), (M=1, C=0), (M=1, C=1)

  (M=0, L=0), (M=0, L=1), (M=1, L=0), (M=1, L=1)

  (C=0, L=0), (C=0, L=1), (C=1, L=0), (C=1, L=1)
- 4 tests pour presque tout couvrir : (0,0,1), (0,1,0), (1,0,1), (1,1,0)

  Avec l'ajout de ces deux tests, tout est couvert : (1,0,0) et (1,1,1)

La Rochelle Université

GRAPHE DE CONTRÔLE

Sommets :

blocs élémentaires du programme, ou prédicats des conditionnelles/boucles

Bloc élémentaire : séquence d'instructions

- en partant de la première instruction
- en exécutant toutes les autres instructions

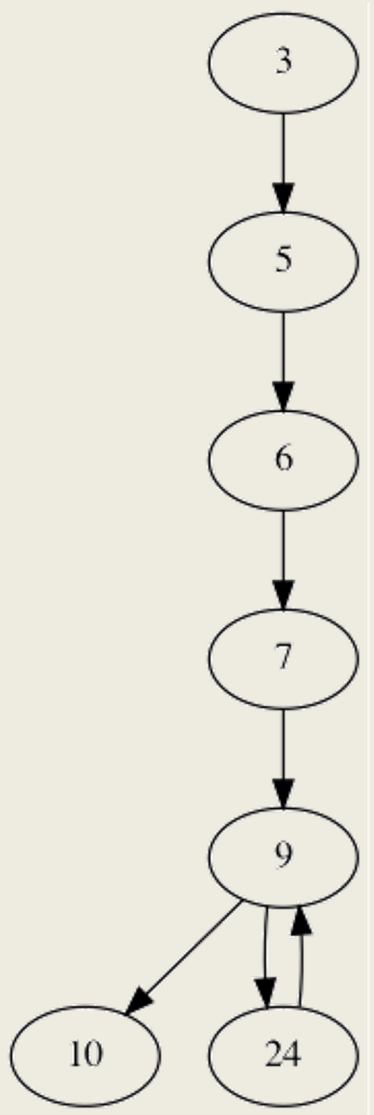
Arcs : enchaînements d'exécution possibles entre deux sommets

Un point d'entrée E et un point de sortie S

La Rochelle Université

Graphe de contrôle - Exemple

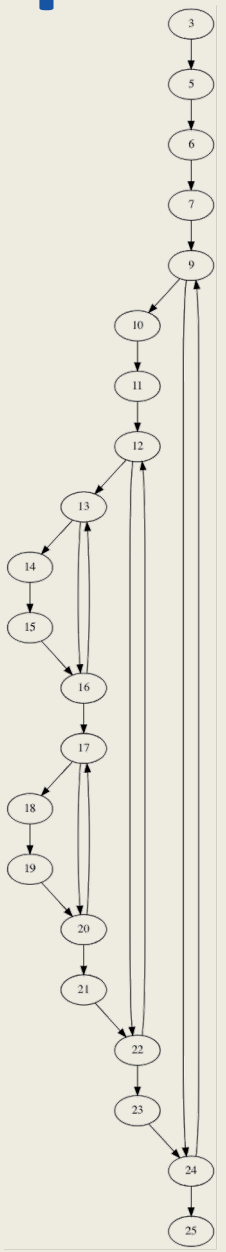
```
1   public class MonProg8a{
2
3       public static void main(String[] args){
4
5           Tortue rosalie = new Tortue();
6           int y = 100;
7           int x;
8
9           while (y<=550){                          // plusieurs créneaux
10              rosalie.saute(50,y);
11              x = rosalie.position().x();
12             while(x<550){                         // pour dessiner un créneau
13                 for (int i=0; i<2; i++){          // premier demi-créneau
14                     rosalie.avance(50);
15                     rosalie.tourneDroite(90);
16                 }
17                 for (int i=0; i<2; i++){          // deuxième demi-créneau
18                     rosalie.avance(50);
19                     rosalie.tourneGauche(90);
20                 }
21                 x = rosalie.position().x();
22             }
23             y+=100;
24          }
25      }
26  }
```



La Rochelle Université

Graphe de contrôle - Exemple

```
1   public class MonProg8a{
2
3       public static void main(String[] args){
4
5           Tortue rosalie = new Tortue();
6           int y = 100;
7           int x;
8
9           while (y<=550){                          // plusieurs créneaux
10              rosalie.saute(50,y);
11              x = rosalie.position().x();
12             while(x<550){                         // pour dessiner un créneau
13                 for (int i=0; i<2; i++){          // premier demi-créneau
14                     rosalie.avance(50);
15                     rosalie.tourneDroite(90);
16                  }
17                 for (int i=0; i<2; i++){          // deuxième demi-créneau
18                     rosalie.avance(50);
19                     rosalie.tourneGauche(90);
20                  }
21                  x = rosalie.position().x();
22              }
23              y+=100;
24          }
25      }
26  }
```



La Rochelle Université

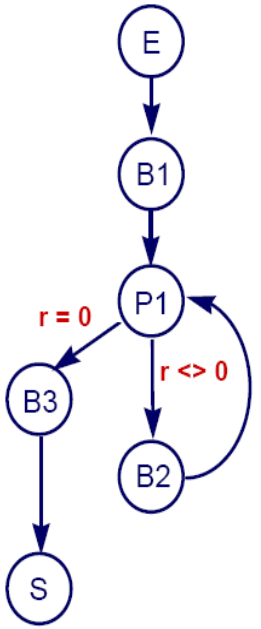
ECRIRE LES TESTS A PARTIR DU GRAPHE DE CONTRÔLE

Analyse de la couverture du flot de contrôle

- couverture des noeuds (instructions)
- couverture des arcs (branches)
- Couverture des chemins

Précondition : p0 >= q0 > 0
Effet : q = pgcd(p0, q0)

```
read(p, q) ;                    B1
r:= p - p // q * q ;
while  r <> 0  do               P1
begin
    p := q;                     B2
    q := r;
    r := p - p // q * q
end ;
write(q) ;                      B3
```



NOMBRE CYCLOMATIQUE

À partir du graphe de contrôle on peut calculer le nombre cyclomatique (Mc Cabe) nMC tel que :

- nMC = nbArcs – nbNoeuds + 2

Le nombre cyclomatique représente :

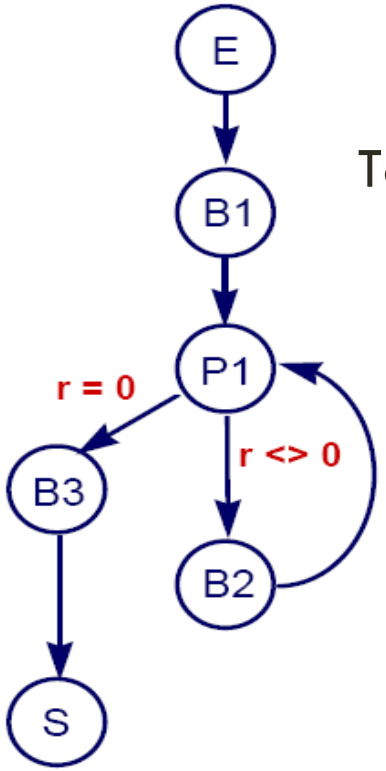
- Le nombre de chemins linéairement indépendants dans un graphe fortement connexe
- Le nombre de décisions + 1 prises dans un programme

Evaluation du nombre de tests minimaux

La Rochelle Université

Précondition : p0 >= q0 > 0
Effet : q = pgcd(p0, q0)

```
read(p, q) ;                    B1
r:= p - p // q * q ;
while  r <> 0  do               P1
begin
    p := q;                     B2
    q := r;
    r := p - p // q * q
end ;
write(q) ;                      B3
```



Nb Mac Cabe = 2

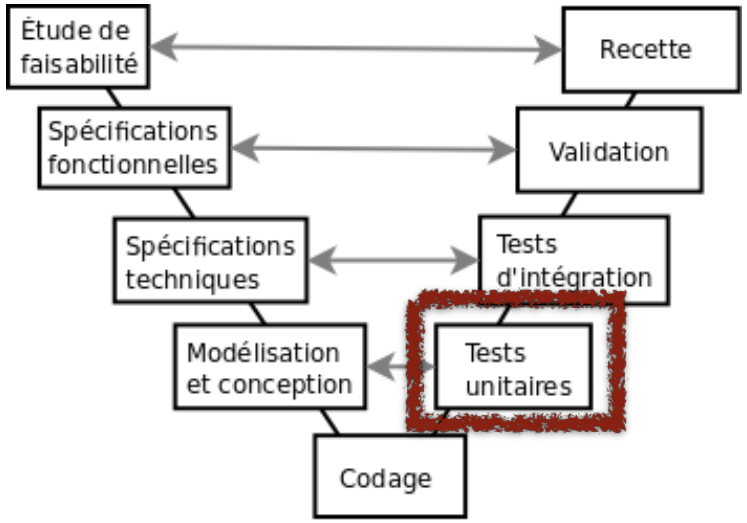
Chemins :

- 1 : E - B1 - P1 - B2 - P1 - B3 - S
- 2 : E - B1 - P1 - B3 - S

Tests sensibilisant les chemins

- 1 : 25 - 10
- 2 : 12 - 4

2/ COUVERTURE DU CODE



Couverture :

A. Les instructions

B. Les branches

C. Les chemins

D. Les conditions multiples

Exemple : type d'un triangle

Paramètre du programme : a,b,c : int

Retour : le type de triangle (scalene, isocèle, nonTriangle, ~~entréesInvalides~~)

```
1   read a,b,c
2   type = "scalene"
3   if (a==b || b==c || a=c)
4       type = "isocele"
5   if (a==b && b==c)
6       type = "equilateral"
7   if (a>=b+c || b>=a+c || c>=a+b)
8       type = "nonTriangle"
9   if (a<=0 || b<=0 || c<=0)
10      type = "entreesInvalides"
11  print type
```

A/ COUVERTURE « TOUTES LES INSTRUCTIONS »

		3,4,5	3,5,3	0,1,0	4,4,4
1	read a,b,c	x	x	x	x
2	type = "scalene"	x	x	x	x
3	if (a==b || b==c || a=c)	x	x	x	x
4	type = "isocele"		x	x	x
5	if (a==b && b==c)	x	x	x	x
6	type = "equilateral"				x
7	if (a>=b+c || b>=a+c || c>=a+b)	x	x	x	x
8	type = "nonTriangle"			x	
9	if (a<=0 || b<=0 || c<=0)	x	x	x	x
10	type = "entreesInvalides"			x	
11	print type	x	x	x	x
	Résultat attendu (RA)	**Scalene**	**Isocele**	**Entrée invalide**	**Equilateral**

La Rochelle Université

C/ COUVERTURE « TOUTES LES CHEMINS»

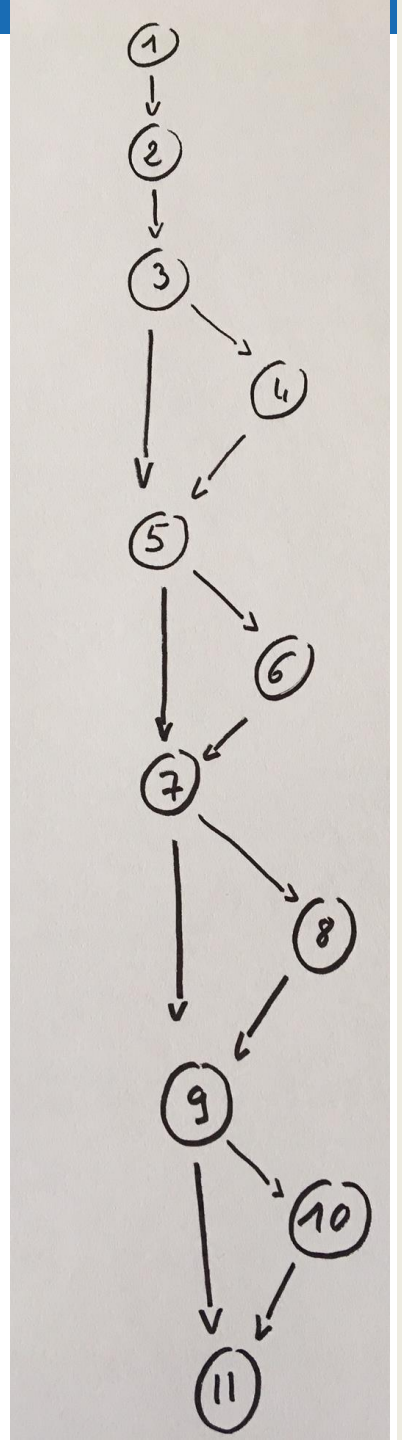
Nombre de chemins = nombre de Mc Cabe

A-N + 2

Exemple :

- 14 - 11 +2 = 5

- Liste des chemins :
  - 1-2-3-5-7-9-11
  - 1-2-3-4-5-7-9-11
  - 1-2-3-5-6–7-9-11
  - 1-2-3-5-7-8–9-11
  - 1-2-3-5-7-9-10–11



C/ COUVERTURE « TOUTES LES CHEMINS»

		DT	Résultat attendu
1	1-2-3-5-7-9-11	3,4,5	Scalène
2	1-2-3-4-5-7-9-11	5,8,5	Isocèle
3	1-2-3-5-6–7-9-11	5,5,5	Equilatéral
4	1-2-3-5-7-8–9-11	9,2,2	NonTriangle
5	1-2-3-5-7-9-10–11	0,3,5	Entrées Invalides

```
1   read a,b,c
2   type = "scalene"
3   if (a==b || b==c || a=c)
4       type = "isocele"
5   if (a==b && b==c)
6       type = "equilateral"
7   if (a>=b+c || b>=a+c || c>=a+b)
8       type = "nonTriangle"
9   if (a<=0 || b<=0 || c<=0)
10      type = "entreesInvalides"
11  print type
```

CHEMINS

- 1-2-3-5-7-9-11
- 1-2-3-4-5-7-9-11
- 1-2-3-5-6–7-9-11
- 1-2-3-5-7-8–9-11
- 1-2-3-5-7-9-10–11

La Rochelle Université

Source

- Cours d’Armelle Prigent

La Rochelle Université