Test logiciel

Objectif et plan du du cours

- Présenter les concepts de base sur le test logiciel

- Introduire des techniques simples pour construire des tests
  - A partir de la spécification informelle du programme
  - A partir de l'analyse du code

- Introduire un outil simple d'exécution de tests pour java: junit

Test logiciel

Concepts de base

Ioannis Parissis – UFR IMA – Laboratoire LIG

Définition du test

- Le test est un moyen :
  - de montrer qu'un programme est correct?
    - Non, en théorie, mais seul moyen de vérifier le fonctionnement du logiciel avant livraison
  - de montrer qu'un programme n'est pas correct?
    - Certainement (pas correct => contient des défauts)
  - de trouver et de corriger les défauts d'un programme?
    - Non, test ≠ mise au point (déboguage)

Définitions

Tester un logiciel consiste à l'exécuter en ayant la totale maîtrise ***des données qui lui sont fournies en entrée (jeux de test)*** *tout en* ***vérifiant que son comportement est celui attendu***

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)

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

Le test est une technique de contrôle consistant à s'assurer, au moyen de son exécution, que le comportement d'un programme est ***conforme à des données préétablies***.
(AFCIQ)

Les 6 facettes du test
(ou les 6 questions qu'on doit se poser quand on teste)

- **Pourquoi ?**
  - –
- **Comment ?**
  - –
- **Combien ?**
  - –
- **Quoi ?**
  - –
- **Où ?**
  - –
- **Quand ?**
  - –

Les 6 facettes du test
(ou les 6 questions qu'on doit se poser quand on teste)

- **Pourquoi ?**
  – *Objectif*
- **Comment ?**
  –
- **Combien ?**
  –
- **Quoi ?**
  –
- **Où ?**
  –
- **Quand ?**
  –

Conformité (cf. protocoles),
recherche défauts,
évaluation utilisabilité
…

Les 6 facettes du test
(ou les 6 questions qu'on doit se poser quand on teste)

- **Pourquoi ?**
  - –
- **Comment ?**
  - – *Sélection*
- **Combien ?**
  - –
- **Quoi ?**
  - –
- **Où ?**
  - –
- **Quand ?**
  - –

Aléatoire, déterministe, statistique, manuelle, systématique, automatique …

Les 6 facettes du test
(ou les 6 questions qu'on doit se poser quand on teste)

- **Pourquoi ?**
  –
- **Comment ?**
  –
- **Combien ?**
  – *Arrêt*
- **Quoi ?**
  –
- **Où ?**
  –
- **Quand ?**
  –

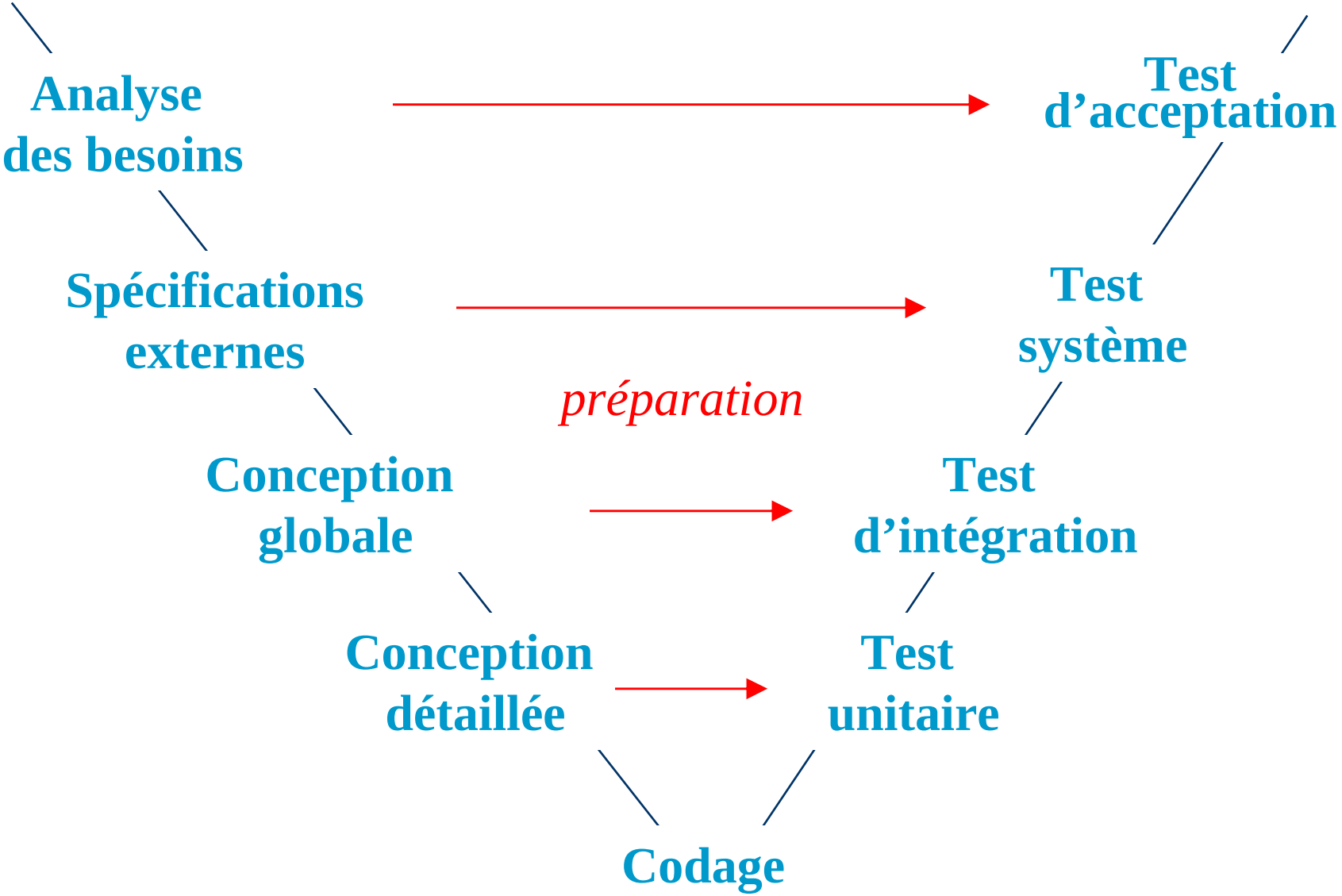
Critères d'adéquation (couverture…), analyse de fiabilité

Les 6 facettes du test
(ou les 6 questions qu'on doit se poser quand on teste)

- **Pourquoi ?**
  - –
- **Comment ?**
  - –
- **Combien ?**
  - –
- **Quoi ?**
  - – *Niveau*
- **Où ?**
  - –
- **Quand ?**
  - –

Test unitaire, d'intégration, système…

Niveaux de test



Niveaux de test

- Test unitaire
  - Unité du logiciel ?
    - Classe
    - Méthode
    - Fonction
    - Module
  - Utilisation de méthodes standard
  - Multitude de techniques et outils
- Test d'intégration
  - Choix d'une stratégie d'intégration
  - Tester les interactions entre unités testées individuellement au préalable
- Test système
  - Vérifier la conformité aux spécifications
  - Effectué avant installation chez le client
  - Dérivé du dossier des spécifications et du plan de test d'acceptation
- Test d'acceptation (recette)
  - Effectué chez le client

Les 6 facettes du test
(ou les 6 questions qu'on doit se poser quand on teste)

- **Pourquoi ?**
  –
- **Comment ?**
  –
- **Combien ?**
  –
- **Quoi ?**
  –
- **Où ?**
  – *Environnement*
- **Quand ?**
  –

Réel, simulé
(cf. log. embarqués)…

Les 6 facettes du test
(ou les 6 questions qu'on doit se poser quand on teste)

- **Pourquoi ?**
  –
- **Comment ?**
  –
- **Combien ?**
  –
- **Quoi ?**
  –
- **Où ?**
  –
- **Quand ?**
  – *Avant (asap) ou après déploiement*

Évolution de la perception du test dans le temps

- Années 50
  - test = mise au point
- 1960 - 1980
  - test = «démonstration» (preuve de correction)
- 1980
  - test = «destruction»
    - « détruire » = trouver des défauts
- Aujourd'hui
  - test = «évaluation»
    - moyen d'évaluation de la qualité

Faute, erreur, défaillance

- Faute ou défaut
  - Cause supposée ou adjugée d'une erreur
- Erreur
  - Etat du système susceptible de provoquer une défaillance
- Défaillance
  - Le service fourni dévie de l'accomplissement de la fonction du système
  - *Le test a pour objectif de mettre en évidence des défaillances*

Exemple

```
T : tablea
icour <- 1
imax <- 1
Tant que icour <= n
    Si T[icour] > T[imax] alors imax <- icour
    icour<-icour+1
Afficher T[imax]
```

Je suis une faute (parfois on m'appelle défaut)

T = [1 2 3]

pas d'erreur, pas de défaillance

T = [2 1 3]

erreur, pas de défaillance

T = [3 2 1]

défaillance

Caractérisation du test

- Quelle que soit la méthode de test utilisée, on y distingue *trois étapes*
  - **Sélection (génération)** des jeux de test
    - **Critères de sélection**
  - **Exécution**
  - **Observation** des résultats

Sélection

- Programme : application d'un domaine d'entrée E vers un domaine de sortie S
  - *Domaines de taille infinie en pratique*
    - Exemple: domaine d'entrée d'un traitement de texte?
- Sélection d'un sous domaine T(E) de E suivant *un critère*
  - Critère de **sélection**
    - le critère sert à la construction des jeux de test.
  - Critère d'**arrêt** ou d'**adéquation**
    - le critère permet d'affirmer qu'un jeu de test (arbitraire) est « bon ».
- Test **exhaustif** (cas rare)
  - E = T(E)

Exécution

- Fourniture des entrées au programme
  - Données entrées manuellement
    - peut-on automatiser?
  - Données complexes ou nombreuses (trajectoires d'un missile, variation d'altitude d'un avion, …)
    - prévoir des simulateurs/moniteurs
- Récupération des sorties
  - Volume important
    - base de données
  - Enregistrer toute information utile à l'analyse des résultats
    - charge processeur
    - occupation mémoire
    - …

Observation : « problème de l'oracle »

- Objectif : répondre à «le programme a-t-il fourni le bon résultat?»
  - Observation effectuée
    - Pendant l'exécution
      - détection des défaillances au moment où elles se produisent
    - Après l'exécution
      - analyse des traces de l'exécution
  - Observation par un humain ou automatique
    - Observation automatique : est-ce possible et à quel prix?
- Oracle automatique
  - En théorie, l'oracle parfait ne peut pas exister
    - Oracle parfait = programme équivalent à celui qu'on teste (indécidabilité)
  - En pratique : oracle approximatif
    - Table associant certaines entrées aux sorties attendues
    - Propriétés essentielles (« l'avion ne s'écrasera pas »)

Test logiciel

Techniques simples de test unitaire

Ioannis Parissis – UFR IMA – Laboratoire LIG

Types de test unitaire

- Classification selon les *modèles* utilisés définir des critères de sélection ou d'adéquation
  - Modèles de spécifications
    - Conformité du programme à ses spécifications
  - Modèles de la structure du programme
    - Couverture de le structure
  - Modèles de fautes
    - Couverture des fautes possibles
  - Modèles aléatoires et statistiques du domaine d'entrée
    - Utilisés pour la génération de données en combinaison avec d'autres modèles

- Autre classification : boîte noire - boîte de verre (cf. test de circuits)
  - Test en « boîte noire »
    - On ne connaît du programme qu'une spécification (informelle ou formelle)
    - On ignore sa structure interne (code)
  - Test en « boîte de verre » (ou « boîte blanche »)
    - Le code du programme est connu et utilisé pour le test

Modèles de spécifications

- Spécifications informelles
  - Textes en langue naturelle
  - Les plus utilisées dans le « monde réel »
- Spécifications semi-formelles
  - Diagrammes, schémas formalisant certains aspects de la spécification mais sans sémantique formelle
    - UML : Diagrammes de classes UML, diagrammes de cas d'utilisation…
  - Percée significative avec UML et la mouvance « Model-driven … »
- Spécifications formelles
  - Spécification de comportement à l'aide de systèmes de transitions, spécifications algébriques…
    - Machines à états, CSP, UML …
  - Approches dont la portée est plus restreinte mais qui ne cessent de gagner du terrain

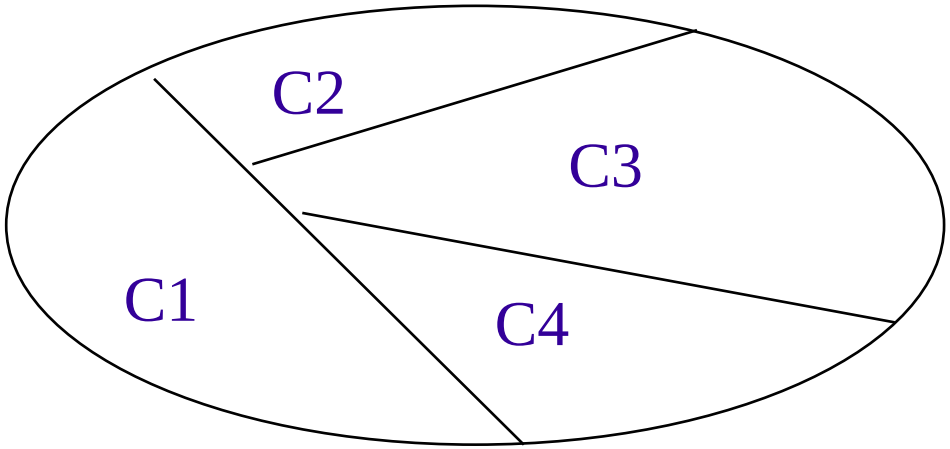
Techniques de test basées sur des modèles informels des spécifications

- Déterminer à partir des spécifications
  - Domaine de définition
  - Domaine de valeurs
  - Comportement attendu
    - Résultats attendus
    - Propriétés

- Techniques connues:
  - Partitionnement en classes d'équivalence
  - Étude des « cas limites »
  - Méthode de Catégories et Partitions
  - Méthode des Graphes Causes - Effets

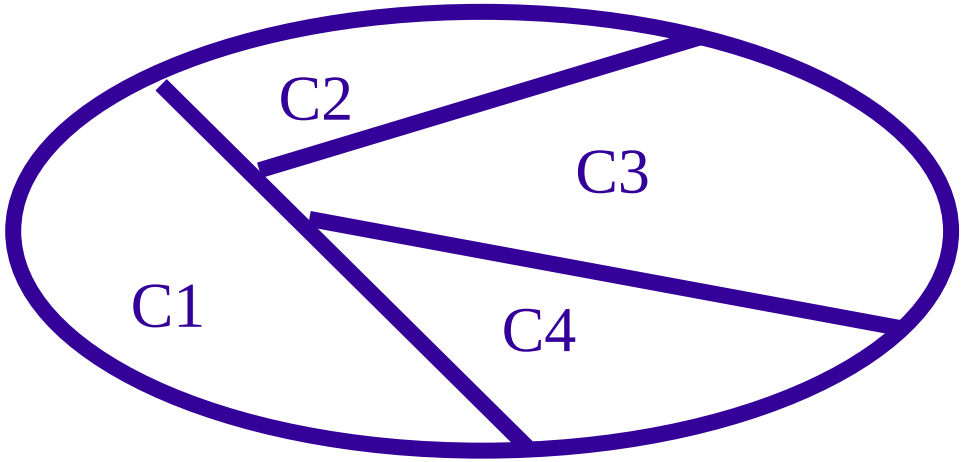
Partitionnement en classes d ’équivalence

- Objectif
  - Sélectionner des valeurs significatives pour les entrées en nombre restreint
- Principe
  - Diviser le domaine d’entrée en classes d’équivalence
    - Tous les éléments d’une classe ont la même probabilité de révéler un défaut
  - Retenir un élément par classe



Étude des cas limites

- Objectif
  - Déterminer les « valeurs aux bornes »
  - Hypothèses
    - les valeurs aux bornes ont une probabilité différente de révéler des fautes
    - révèlent des fautes différentes
- Nouvelles classes d'équivalence
  - Regroupement autour des valeurs aux bornes



Étude des cas limites (suite)

- Objectifs
  - Production systématique de tests
  - Suggestion d'une mesure de couverture de spécifications
- Classes d'équivalence
  - Valeurs particulières des entrées du programme
  - Cas limites
  - Expérience du testeur
- Difficilement automatisable

Méthode des « Catégories et Partitions » (suite)

- Étape 1 : Analyse de la spécification
  - Identification des unités fonctionnelles élémentaires simples, compréhensibles et pouvant être testées séparément
  - Pour chaque unité, identification des :
    - paramètres
    - caractéristiques des paramètres
  - Organisation en **catégories**
    - Chaque catégorie correspond à un ensemble de valeurs d'une caractéristique d'un paramètre
- Étape 2 : Détermination des contraintes de choix
  - Contraintes sur la valeur des caractéristiques de plusieurs paramètres simultanément.
- Étape 3 : création des partitions
  - Combinaison exhaustive entre catégories et contraintes de choix.
- Étape 4 : Sélection des jeux de test
  - Constitution des jeux de test de sorte qu'au moins un jeu par partition soit sélectionné.

Exercice : application de la méthode

- On considère un programme qui lit et imprime une liste de télégrammes en réalisant quelques traitements supplémentaires : suppression des espaces redondants, décompte et impression du nombre total de mots (à l'exception des marqueurs définis ci-dessous), indication de la présence d'au moins un mot de longueur supérieure à M, impression du texte, après traitement, en lignes contenant au plus L caractères (M, L spécifiés par l'utilisateur).

- Les caractères autorisés pour la saisie du télégramme sont les caractères alphanumériques et l'espace. Un mot d'un télégramme est séparé de son suivant par un ou plusieurs espaces. Deux mots sont réservés à des fonctions spécifiques : STOP (marqueur de fin de phrase) et ZZZZ (marqueur de fin de télégramme).

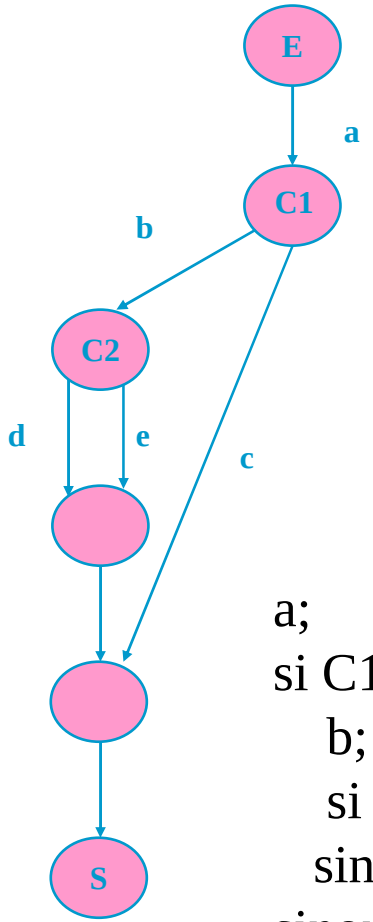
- Le programme traite les télégrammes jusqu'à l'apparition d'un télégramme vide (qui ne contient qu'un seul mot : ZZZZ).

Modèles du code

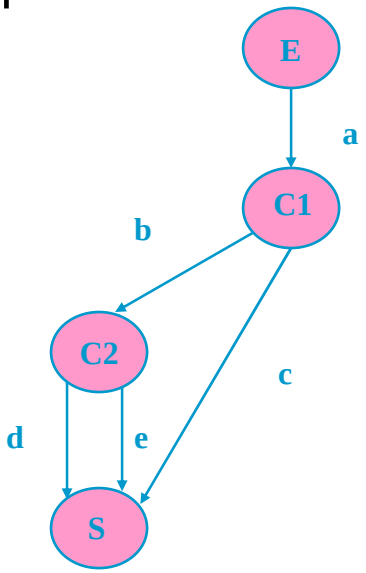
- Graphe de contrôle
  - Bien approprié aux langages impératifs
  - Représentation du flux de contrôle
    - Instructions, branches, chemins
  - Représentation du flux de données
    - Chemins entre définition et utilisation de variables
- Machines à états
  - Correspond à une structure particulière de programmes
  - Représentation du flux de contrôle
    - États, transitions, chemins
- Graphe de flux de données
  - Pour des langages flux de données
    - Exemple : Lustre/SCADE
- Permettent l'automatisation
  - de la mesure de satisfaction des critères définis
  - de la génération des données de test
- Outils industriels disponibles

Graphe de contrôle

- Noeud
  - Saut (in)conditionnel dans le programme
    - Condition de si_alors_sinon, tant_que ...
    - Saut inconditionnel
  - Deux nœuds supplémentaires
    - E (entrée unique)
    - S (sortie unique)
  - Jonction de flux de contrôle
- Branche
  - Arc reliant deux noeuds
  - Suite d'instructions contiguës
- Chemin
  - Suite de branches contiguës



Version complète et optimisée




```
a;
si C1 alors
    b;
    si C2 alors d
   sinon e
sinon c
```

Critères de sélection/adéquation courants définis sur le graphe de contrôle

- Couvertures des instructions
- Couverture des branches
- …
- Couverture des chemins

- Couverture du flux de données

Couverture des instructions

- Objectifs
  - Exécuter au moins une fois toute instruction du programme
  - Détecter des instructions fautives
- Justification – hypothèses
  - Une seule exécution d'une instruction fautive causera une défaillance
- Limites
  - S1;
    if C then S2;
    S3;

    C = vrai satisfait le critère (S1, S2 et S3 exécutés).

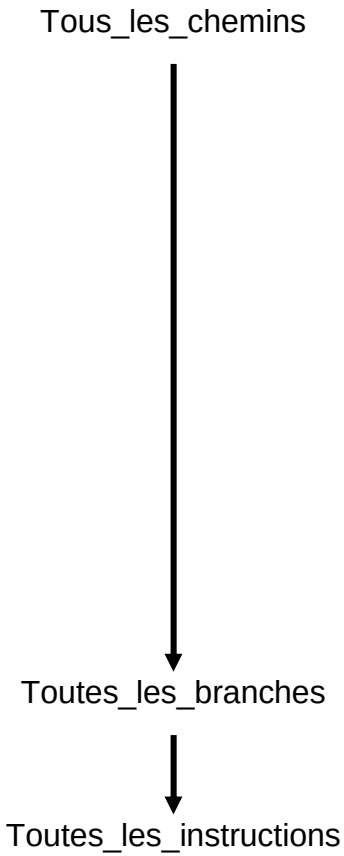
    C = faux potentiellement jamais testé.

Couverture des branches

- Objectifs
  - Exécuter au moins une fois chaque branche
  - Mettre en évidence des défauts dans les instructions conditionnelles ou itératives
- Justification – hypothèses
  - Deux passages par la condition (vrai-faux) suffisent pour mettre en évidence un défaut
- Limites
  - if C1 then S1 else S2;
    if C2 then S3 else S4;

    (C1, C2) = (F, F) et (V, V) satisfont le critère.
    (F, V) et (V, F) jamais testés.

Relation d'inclusion



Exercice

- Quels tests exécuter pour couvrir 100% des instructions (resp. branches) du programme suivant:

```
public String coderTexte ( String texte, int code1, code2 ) {
String texteCode = new String ("");
int pos = 0 ;
int code= code1 ;
if (texte.charAt[0] == 'A')
        code = code2 ;
while (pos < texte.length()){
        if(texte.charAt(pos)!='A')
           texteCode = texteCode +(char)( texte.charAt(pos) + code);
        pos++;
}
return(texteCode);
}
```

Test logiciel

Un outil simple pour l'exécution de tests pour java: junit