Vérification par le test

Jean-Marie Mottu jean-marie.mottu@univ-nantes.fr

Le Traon – Baudry - Sunye

Plan

- Introduction
- Techniques de test
  - Test dynamique
  - Test statique

Le titre : Vérification par le test

- Vérification
  - Contrôler que le programme sous test respecte la spécification
- Test
  - Principe générale : faire des essais
    - Combien ? De quel type ? A quel point ?
  - En opposition avec la preuve qui consiste à formaliser le système pour appliquer des vérifications de niveau mathématique
    - Difficulté de la formalisation

Motivation

- Diminuer le coût d'un logiciel
- Augmenter la qualité
  - Augmenter la confiance

Objectifs

- Le test a pour but de détecter la présence d'erreurs dans un programme vis-à-vis de sa *spécification*
  - éventuellement mise à l'épreuve :
    - de la robustesse
    - des performances
    - de propriétés de sûreté
  - Formalisation de critères pour guider la sélection des tests

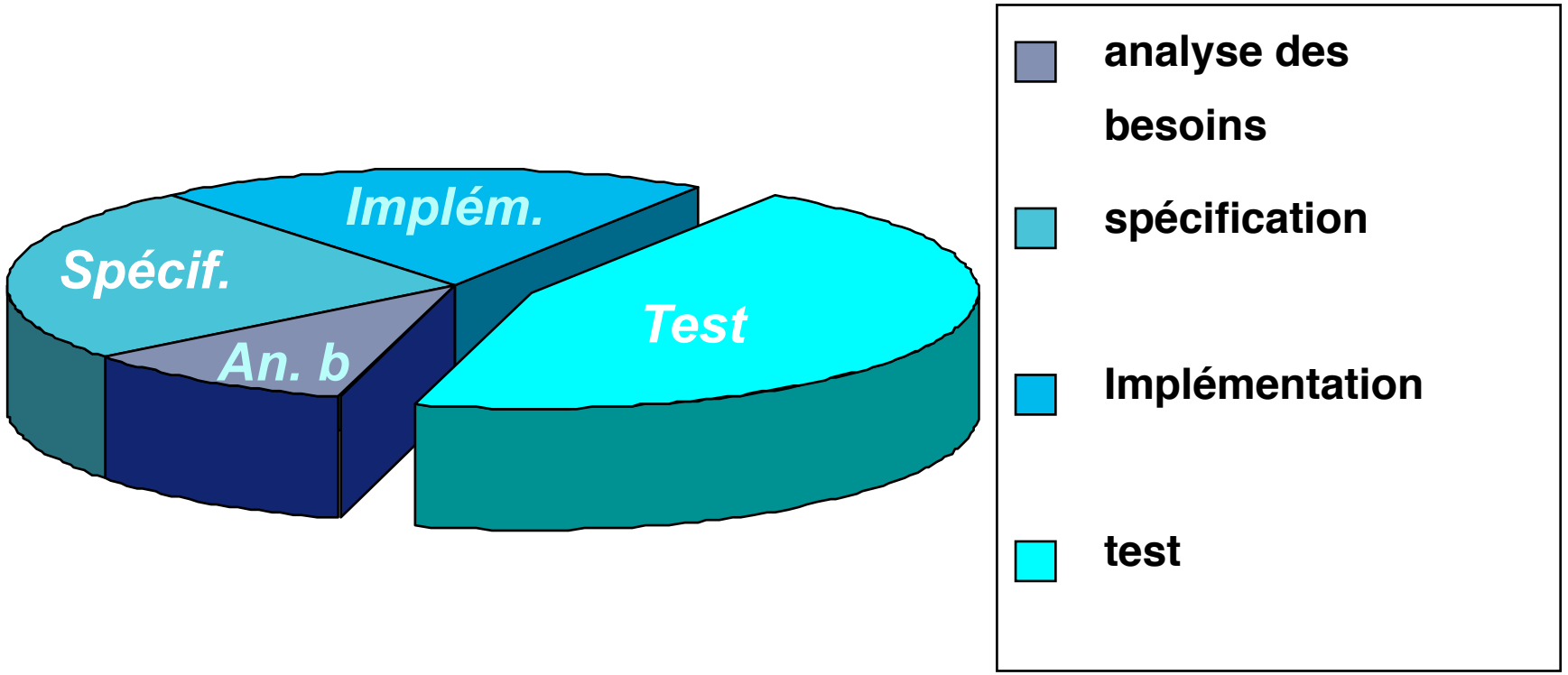
Problématique

- On ne peut pas tester tout le temps ni tous les cas possibles
  - Il faut des critères pour choisir les cas intéressants et la bonne échelle pour le test

- Prouver l'absence d'erreurs dans un programme est un problème indécidable
  - il faut des heuristiques réalistes

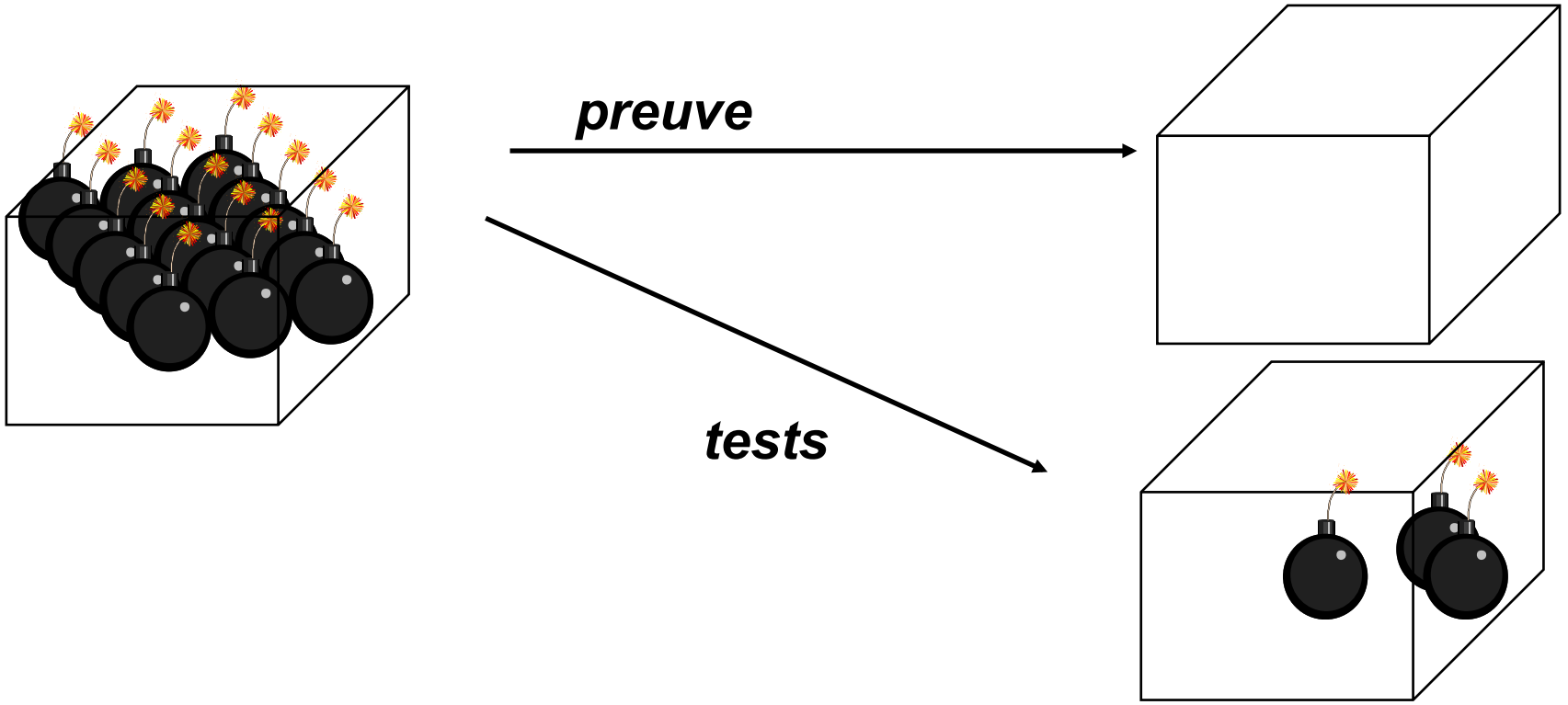
Problématique du test

Le coût du test dans le développement



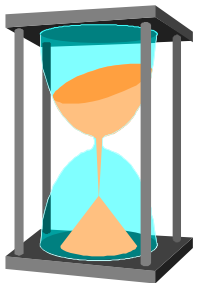
+ maintenance = 80 % du coût global de développement !!!

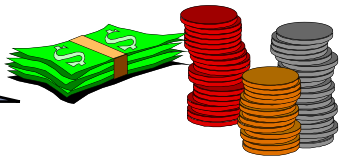
Problématique de la vérification par le test

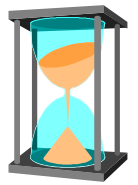


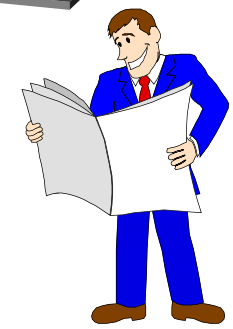
Problématique du test

Pourquoi ?









Problématique d'apprendre à tester

- Un jeune diplômé sur trois commence par faire du test
- La moitié des start-up échouent à cause du trop grand nombre de bugs
  - mauvaise campagne de test
  - maintenance difficile
  - pas de non régression
- Sans technique, le test est extrêmement laborieux

Le test – Définition Générale

Essayer pour observer si ça fonctionne bien.

Apprendre
- pourquoi c'est fait
- ce que ça doit faire
- comment c'est fait
- comment ça marche

Modéliser
S'en faire une idée
Exécuter
Analyser

Qu'y a-t-il à observer?
Que faut-il regarder?
Qu'est-ce qui est visible?
Qu'est ce qu'on cherche?
Comment le regarder?

Qu'est ce qui devrait fonctionner ?
Identifier une erreur
Diagnostiquer une erreur
Catégoriser ces erreurs

Ca peut fonctionner, mais assez vite ?

Qu'est-ce qu'on teste?
(quelles propriétés?)

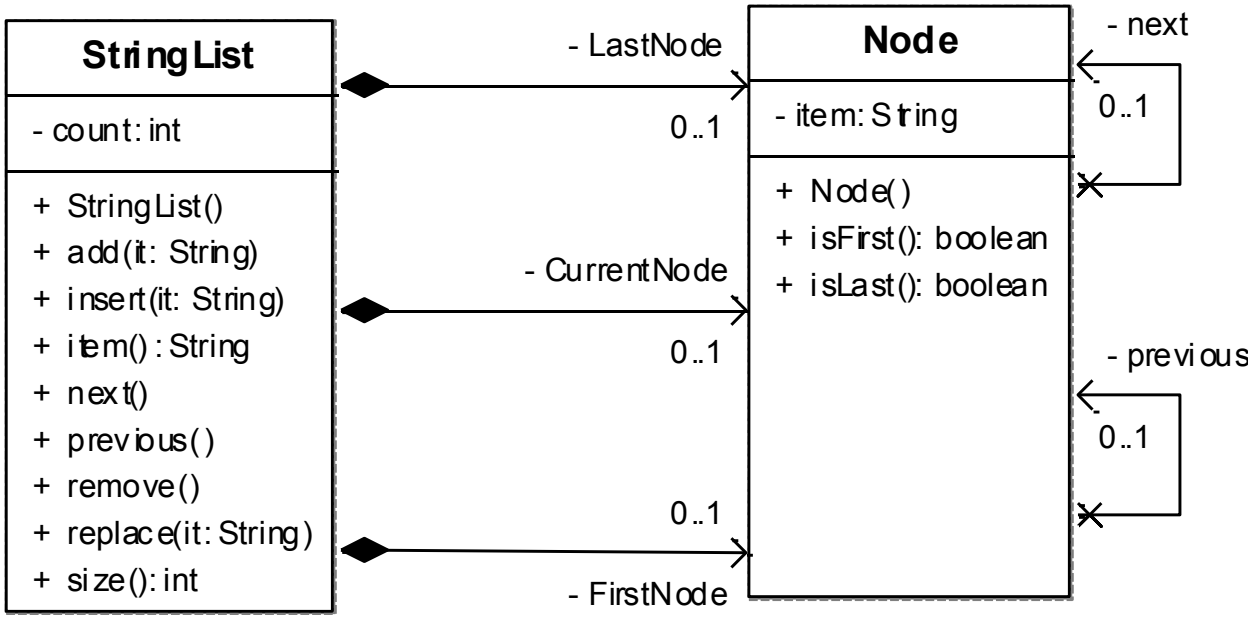
- fonctionnalité
- sécurité / intégrité
- utilisabilité
- cohérence
- maintenabilité
- efficacité
- robustesse
- sûreté de fonctionnement

Test de logiciel

- Plusieurs échelles:
  - Unitaire, intégration, système
- Plusieurs phases
  - non régression / recette
- Plusieurs techniques
  - Dynamique / statique
- Génération de test
  - Fonctionnel / structurel

Exemple

Comment tester la classe StringList?



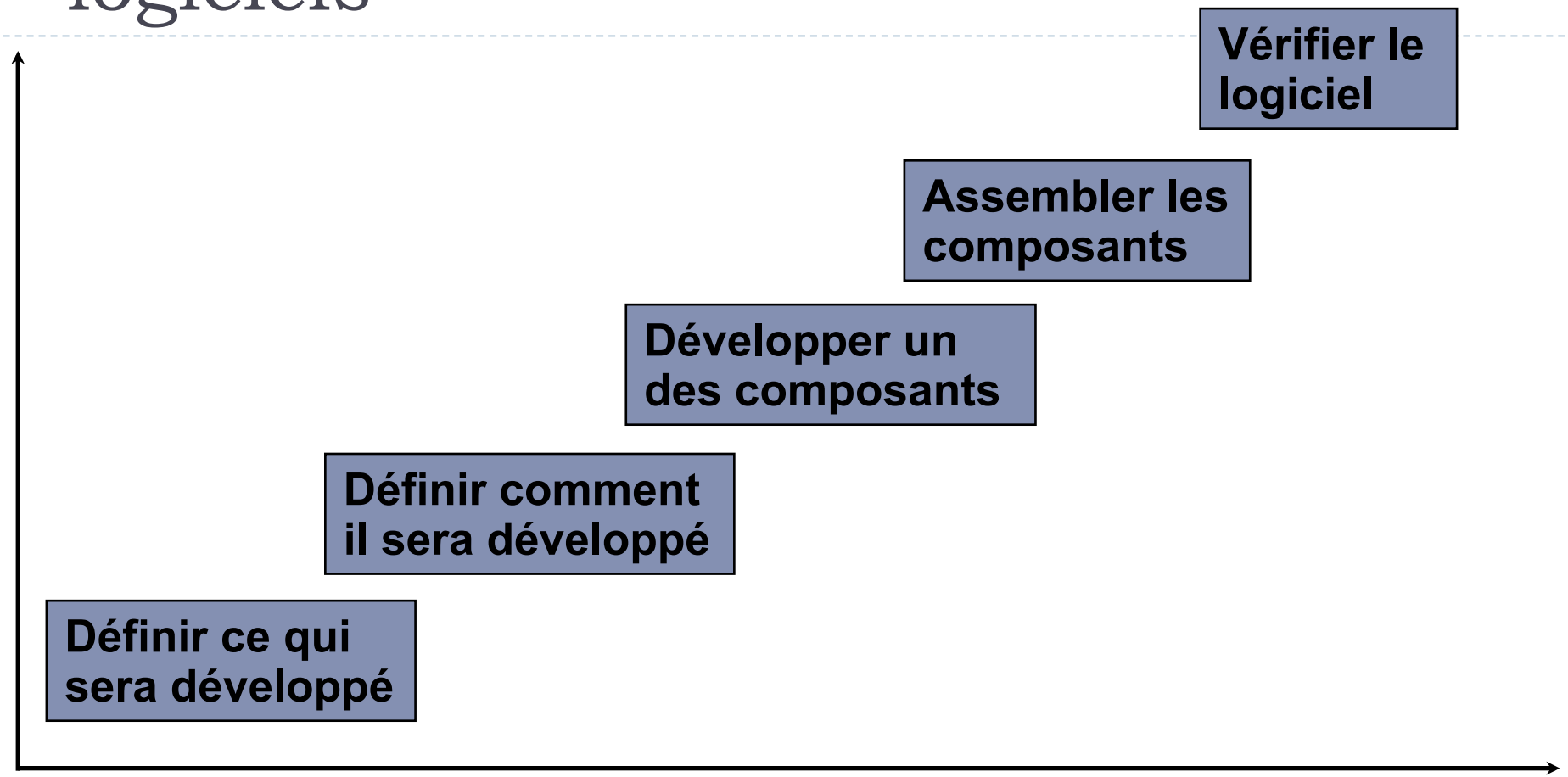
- tester l'ajout dans une liste vide
- tester l'ajout dans une liste avec un élément
- tester le retrait dans une liste avec deux éléments
- ....

Comment écrire ces tests?
Comment les exécuter?
Les tests sont-ils bons?
Est-ce que c'est assez testé?
...

Sur quoi baser la vérification ?

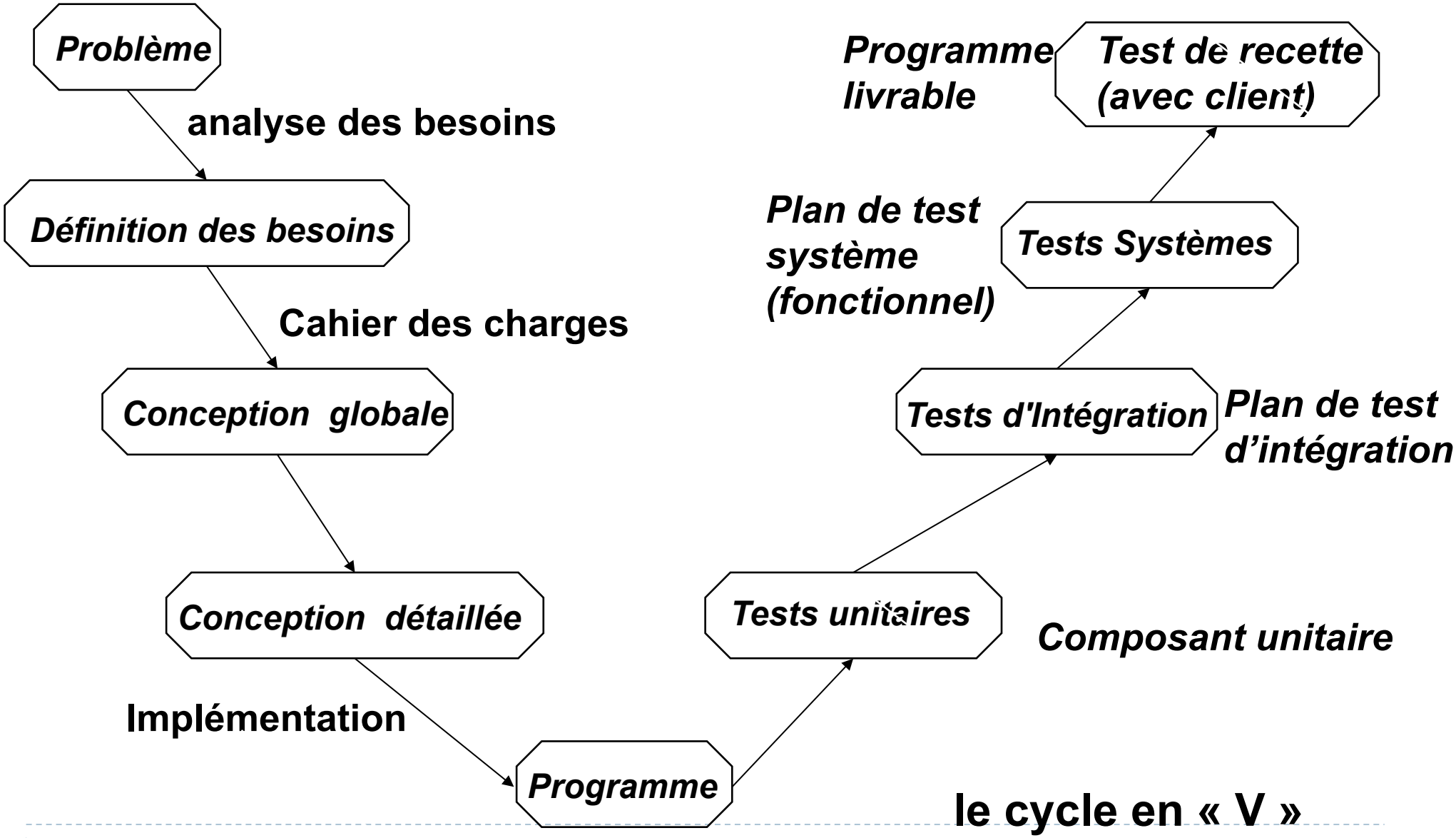
- Une spécification : exprime ce qu'on attend du système
  - un cahier des charges (en langue naturelle)
  - commentaires dans le code
  - contrats sur les opérations (à la Eiffel)
  - un modèle UML
  - une spécification formelle (automate, modèle B...)

Activités du développement de logiciels



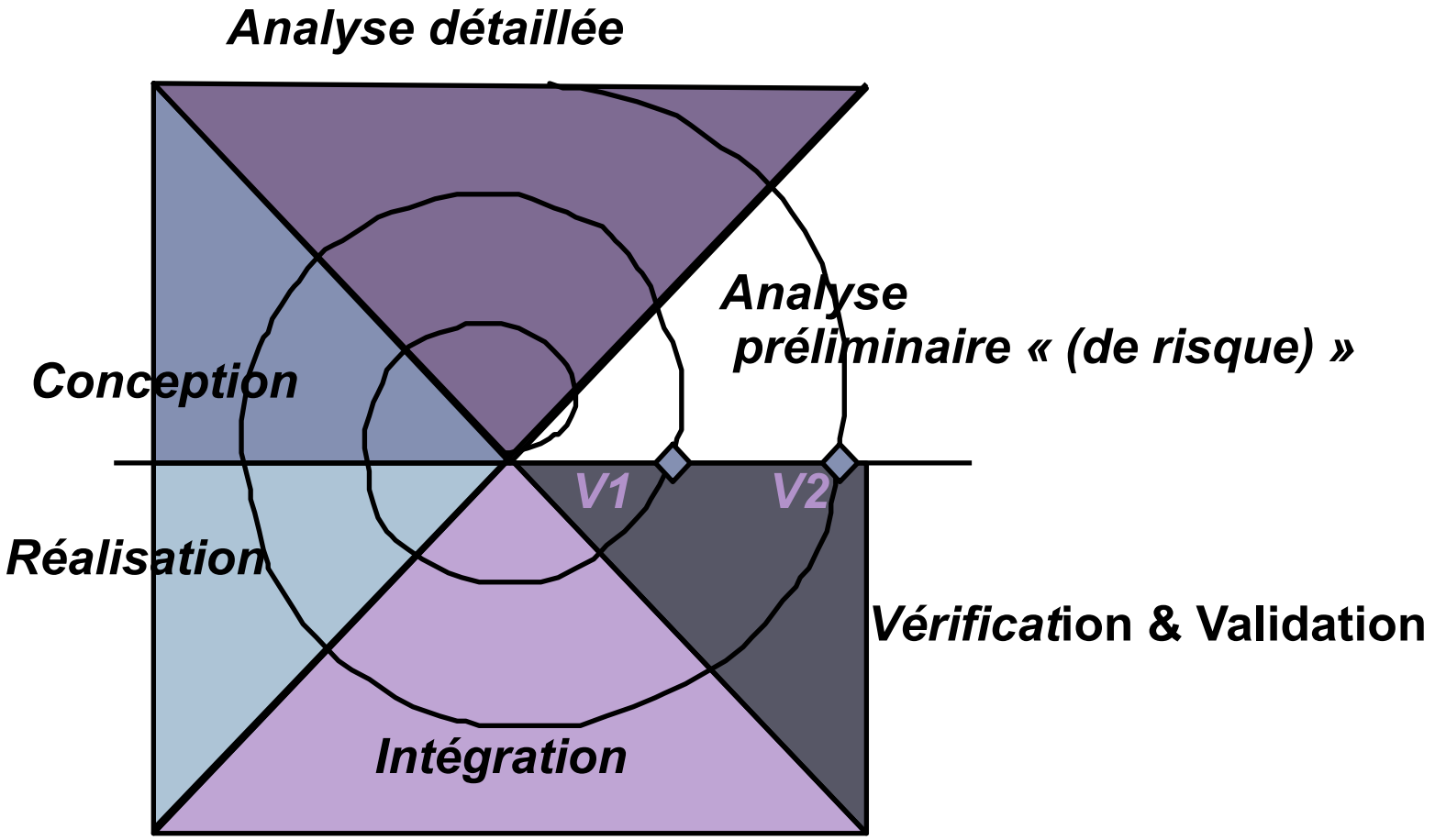
- L'organisation de ces activités et leur enchaînement définit le *cycle de développement* du logiciel

Cycle en V



le cycle en « V »

Cycle de vie en « spirale »



Synergie avec approche par objets

Test unitaire

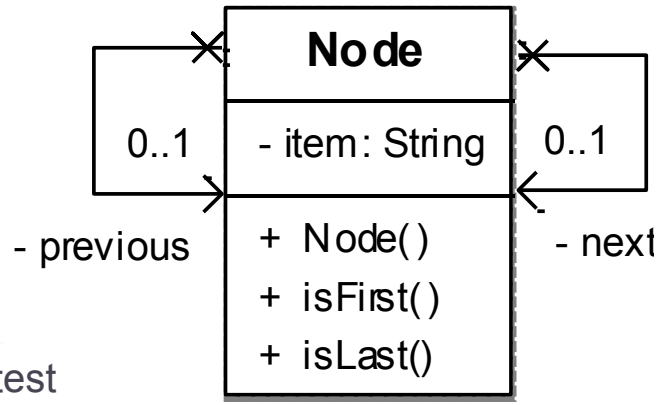
- Vérification d'un module indépendamment des autres
- Vérifier intensivement les fonctions unitaires
- Les unités sont-elles suffisamment spécifiées?
- le code est-il lisible, maintenable...?

Test unitaire

- Pour un langage procédural
  - unité de test = procédure

```
void Ouvrir (char *nom, Compte *C, float S, float D )
{
        C->titulaire = AllouEtCopieNomTitulaire(nom);
        (*C).montant = S ;
        (*C).seuil  = D ;
        (*C).etat = DEJA_OUVERT ;
        (*C).histoire.nbop = 0;
        EnregistrerOperation(C);
        EcrireTexte("Ouverture du compte numero ");
        EcrireEntier(NumeroCourant+1);
        EcrireTexte(", titulaire : \"");
        EcrireTexte(C->titulaire); EcrireCar("");
        ALaLigne();
}
```

- Dans un contexte orienté objet
  - unité de test = classe



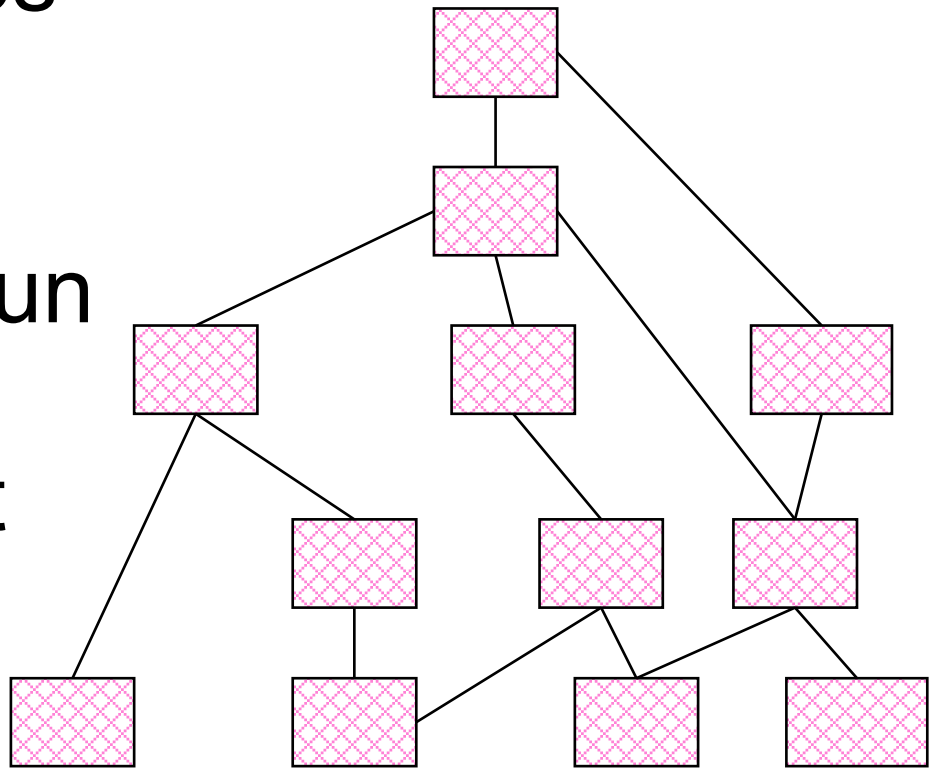
Test d'intégration

- Choisir un ordre pour intégrer et tester les différents modules du système

Test d'intégration

Cas simple: il n'y a pas de cycle dans les dépendances entre modules

Les dépendances forment un arbre et on peut intégrer simplement de bas en haut

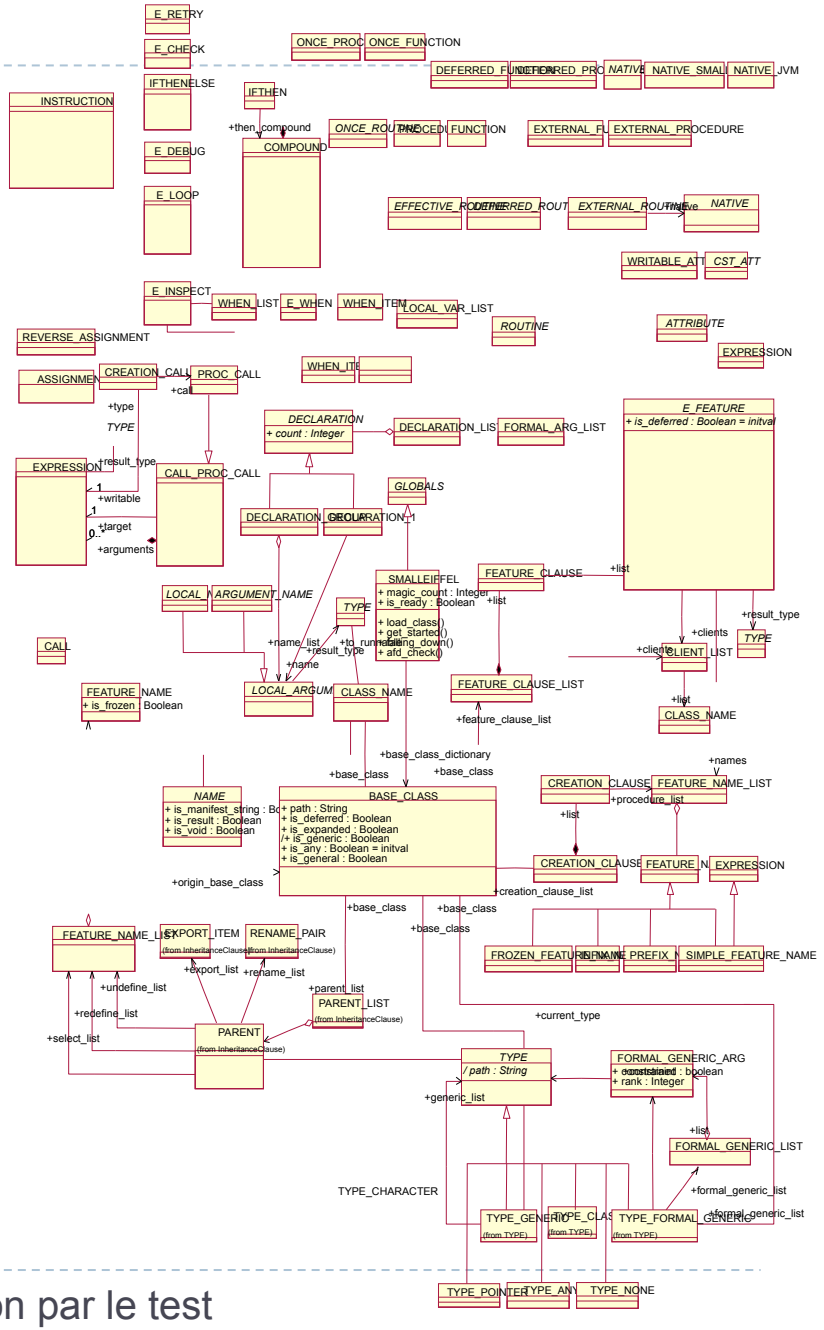


Test d'intégration

Cas plus complexe: il y a des cycles dans les dépendances entre modules

Cas très fréquent dans les systèmes à objets

Il faut des heuristiques pour trouver un ordre d'intégration



Test système

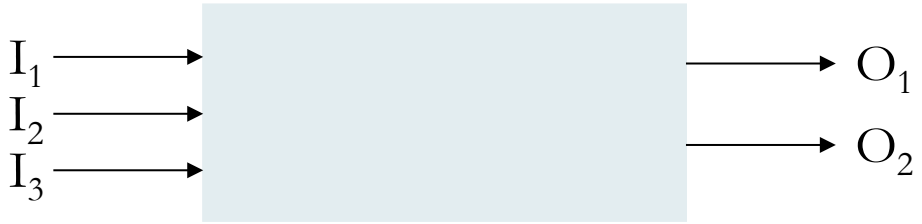
- Valider la globalité du système
  - Les fonctions offertes
  - A partir de l'interface

Test de non régression

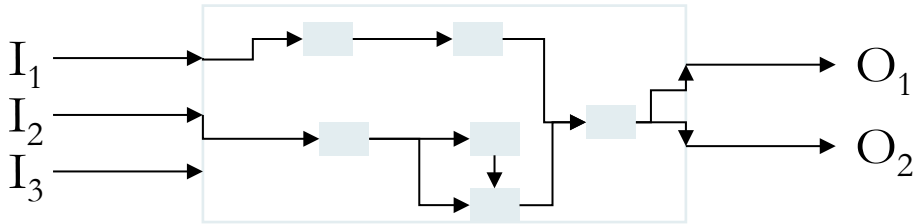
- Consiste à vérifier que des modifications apportées au logiciel n'ont pas introduit de nouvelle erreur
  - vérifier que ce qui marchait marche encore
- Dans la phase de maintenance du logiciel
  - Après refactoring, ajout/suppression de fonctionnalités
- Après la correction d'une faute

La génération de test

- Test fonctionnel (test boîte noire)
  - Utilise la description des fonctionnalités du programme



- Test structurel (test boîte blanche)
  - Utilise la structure interne du programme



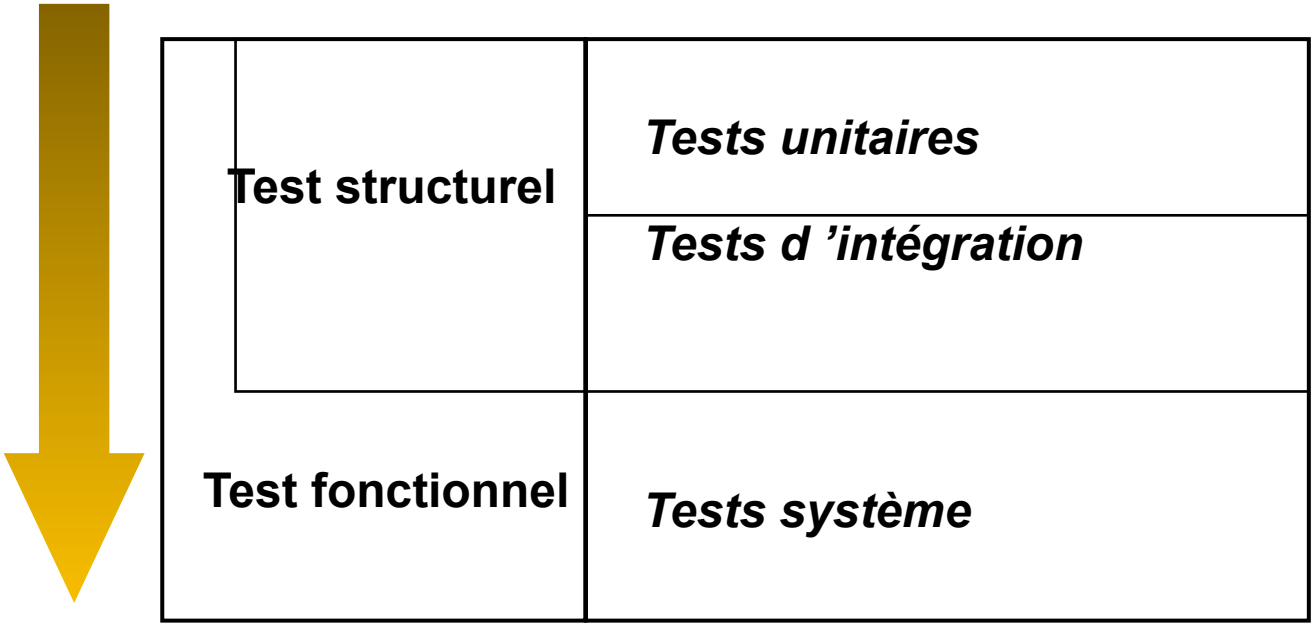
Test fonctionnel

- Spécification formelle
  - Modèle B, Z
  - Automate, système de transitions
- Description en langage naturel
- UML
  - Use cases
  - Diagramme de classes (+ contrats)
  - Machines à états / diagramme de séquence

Test structurel

- A partir d'un modèle du code
  - modèle de contrôle (conditionnelles, boucles...)
  - modèle de données
  - modèle de flot de données (définition, utilisation...)
- Utilisation importante des parcours de graphes
  - critères basés sur la couverture du code

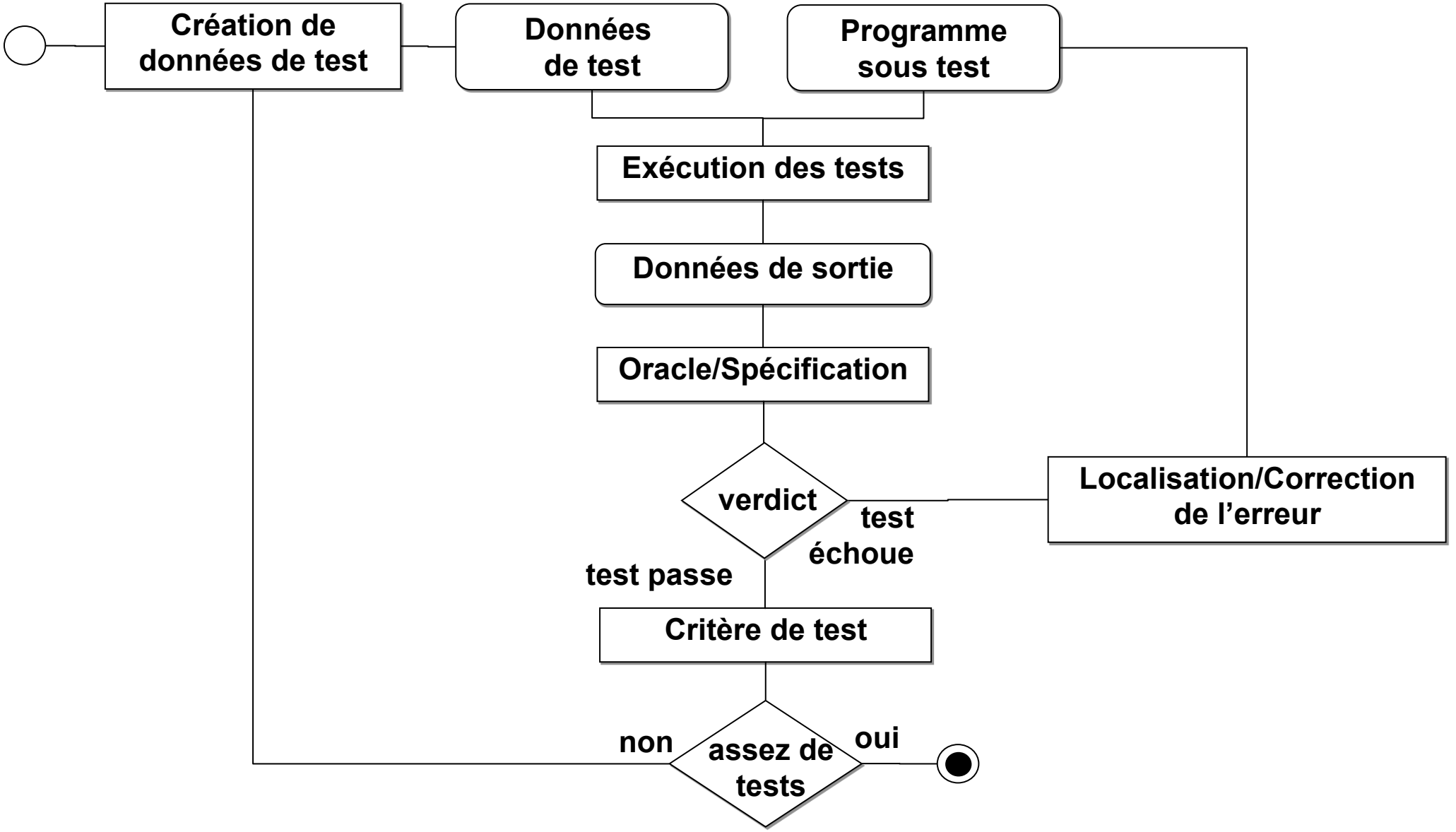
Etapes et hiérarchisation des tests



Quel technique de test ?

- Test statique
  - relecture / revue de code
  - analyse automatique
    - vérification de propriétés, règles de codage...
- Test dynamique
  - on exécute le programme avec des données en entrée et on observe le comportement

Processus du test dynamique



Des données, des activités

- Données
  - Données de test en entrée du programme sous test
  - Données de sortie produites par le programme
  - Verdict

Des données, des activités

- Activités
  - Création des données de test
    - Génération
    - Qualification
  - Oracle
    - Analyse des données de sortie
    - Confrontation avec la spécification
  - Critère de test
    - A-t-on assez testé ?
  - Localisation/correction des erreurs

Construction de test

- Construction déterministe
  - « à la main »
- Génération automatique aléatoire
- Construction automatique aléatoire contrainte
  - mutation
  - test statistique
- Génération automatique guidée par les contraintes

- Reste à savoir quand on a suffisamment testé
  - critères de test structurels, fonctionnels
  - analyse de mutation
- Choisir le bon niveau pour le test

Techniques de test
Test statique

Jean-Marie Mottu
Le Traon – Baudry - Sunye

Test statique

- Ne requiert pas l ’exécution du logiciel sous-test sur des données réelles
- Plusieurs approches
  - inspection de code (lisibilité du code, spécifications complètes...)
  - mesures statiques (couplage, nombre d’imbrications...)

Inspection de code

- Réunions de 4 personnes environ pour inspecter le code
  - 1 modérateur, le programmeur, le concepteur et 1 inspecteur
- Déroulement
  - le programmeur lit et explique son programme
  - le concepteur et l 'inspecteur apportent leur expertise
  - les fautes sont listées
  - (pas corrigées-> à la charge du programmeur)

Inspection de code

- Efficacité : plus de 50 % de l ’ensemble des fautes d ’un projet sont détectées lors des inspections si il y en a (en moyenne plus de 75%)
- Défaut : mise en place lourde, nécessité de lien transversaux entre équipes, risques de tension…tâche plutôt fastidieuse

Test statique

- Règles
  - être méthodique (cf. transparents suivants)
  - un critère : le programme peut-il être repris par quelqu'un qui ne l'a pas fait
  - un second critère : les algorithmes/l'architecture de contrôle apparaît-elle clairement ?
  - décortiquer chaque algo et noter toute redondance curieuse (coller) et toute discontinuité lorsqu'il y a symétrie (ce qui peut révéler une modif incomplète du programme)

Test statique

- Exemple: vérification de la clarté
  - R1 :Détermination des paramètres globaux et de leur impact sur les fonctions propres

```
program recherche_tricho;
uses crt;
const
   max_elt = 50;
   choix1  =  1;
   choix2  =  2;
   fin     =  3;
type
   Tliste  = array[1..max_elt] of integer;
var
   liste              : Tliste;
   taille, choix, val : integer;
   complex            : integer;
```

- But du programme non exprimé
- Manque de commentaires
- Identificateurs non explicites

Test statique

Exemple: vérification de la clarté

R2 : Existence d'un entête clair pour chaque fonction

```
{-------------------------------------------------------------
  Recherche recursive d'une valeur dans une liste triee
-------------------------------------------------------------}
```

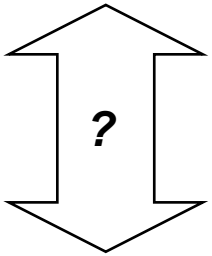
Test statique: pour chaque fonction

Commentaires minimum

manque
- le nom de la fonction
- dépendance avec autres variables/ fonctions

{ parametres d'entree : liste L
la valeur recherchee
indice gauche
indice droit
resultat : complexite de la recherche }

Interface Spécifiée



Interface implantée

```
function rech_rec(L : Tliste ; val, g, d : integer) : integer ;
```

Test statique

```
var i, pt, dt : integer;

begin
   affiche(L, g, d);
   if g<d
      then
         begin
            pt :=g+(d-g) div 3;
            if val > L[pt]
               then
                  begin
                     dt := (pt+1+d) div 2;
                     if val > L[pt]
                        then rech_rec:=2+rech_rec(L, val, dt+1, d)
                        else rech_rec:=2+rech_rec(L, val, pt+1, dt)
                  end
               else rech_rec:=1+rech_rec(L, val, g, pt)
         end
      else rech_rec:=0
end; { rech_rec }
```

Quézako ?

Action non spécifiée

Répétition ?

Test statique

- Métriques
- Analyse d'anomalies
- Preuve
- Exécution symbolique
- Simulation de modèle