Le Test de Logiciel

Reda Bendraou
reda.bendraou{{@}}Lip6.fr
http://pagesperso-systeme.lip6.fr/Reda.Bendraou/

Le contenu de ce support de cours a été influencé par les lectures citées à la fin de ce support.

Plan

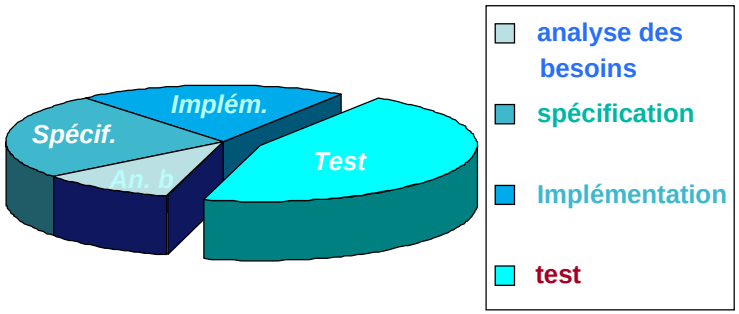
1- Problématique du test

Problématique du test

- 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

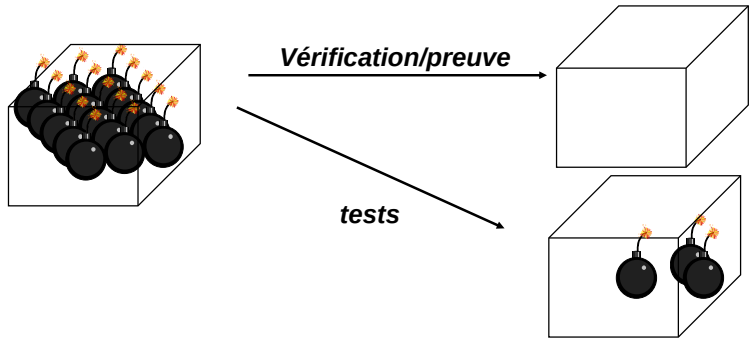
Le coût du test dans le développement



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

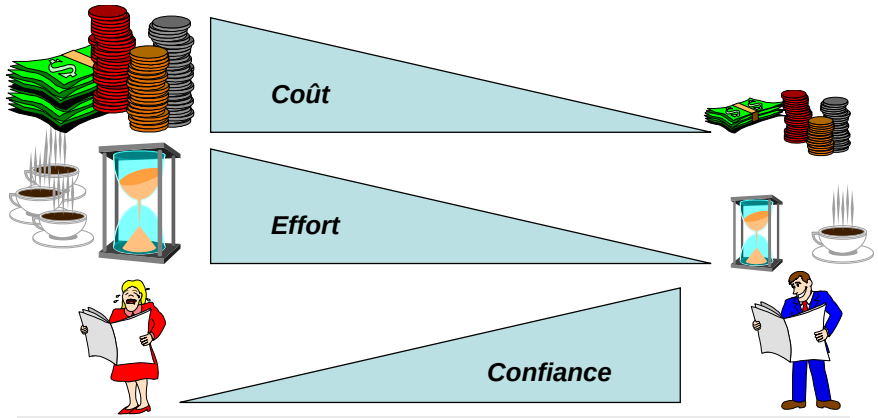
Problématique: une définition !

La validation ⟶ ***conformité du produit par rapport à sa spécification***



Problématique du test

Pourquoi ?



Vocabulaire

Testabilité

faute

erreur ***bogue***

défaillance

Fiabilité (Reliability)

Test de Robustesse

Test statique
Test de non-régression

Tolérance aux fautes

Séquence de test

Données de test

Sûreté de fonctionnement (Dependability)

Jeu de test

Test statistique

Cas de test

Défaillances

- Catastrophe humaine ou financière:
  - Automobile (2004) – régulateur de vitesse
  - Therac-25 (1985-1987) – radiologie et contrôle d'injection de substances radioactives
  - London Ambulance System (1992) – central et dispatch ambulances
  - Iran Air Flight 655 (1988) – guerre d'Irak et missile américain – système radar
  - Ariane 5 (1996)
  - SI du FBI (2005) – SI qui n'a pas pu être déployé
  - Mars Climate Orbiter (1999) – kilos – pounds
  - Bourse de Londres (Taurus, 1993) – SI qui n'a pas pu être déployé
- Image de marque :
  - FT et Bouygues en 2004 – crash des serveurs – indisponibilité 48h

Dues à des bugs

- USS Yorktown (1998)
  - Une division par zéro coupe les moteurs
- Ariane 5 (1996)
  - Mauvaise réutilisation
- Mars orbiter (1999)
  - Plusieurs unités de mesure
- Système de guidage (2002)
  - Initialisation erronée
- The Patriot and the Scud
  - mauvais arrondi dans une soustraction

Dues au processus

- Therac-25 (official report)
  - The software code was not independently reviewed.
  - The software design was not documented with enough detail to support reliability modelling.
  - The system documentation did not adequately explain error codes.
  - AECL personnel were at first dismissive of complaints.
  - The design did not have any hardware interlocks to prevent the electron-beam from operating in its high-energy mode without the target in place.
  - Software from older models had been reused without properly considering the hardware differences.
  - The software assumed that sensors always worked correctly, since there was no way to verify them. (see open loop)
  - Arithmetic overflows could cause the software to bypass safety checks.
  - The software was written in assembly language. While this was more common at the time than it is today, assembly language is harder to debug than high-level languages.
  - ....

Problématique du test

- Un jeune diplômé sur trois commence par faire du test
- 50% des start-up échouent à cause du trop grand nombre de bugs
  - mauvaise campagne de test
  - maintenance difficile
  - pas de non régression

2- Rappels test de logiciel

- Le test: quoi et comment?
- Etapes et processus de test
- Génération de test

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

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

Comment on teste?

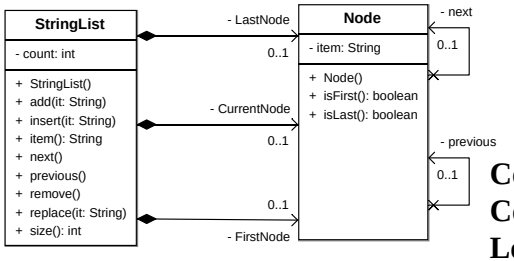
- Test statique
  - relecture / revue de code
  - analyse automatique (vérification de propriétés, règles de codage...
- Test dynamique
  - on exécute le programme avec des valeurs en entrée et on observe le comportement

Avec quoi on teste?

- 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...)

Exemple

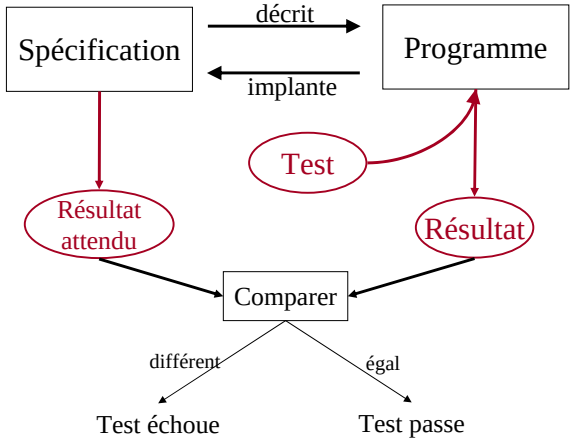
Comment tester la classe StringList?



- 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?
...

Test de logiciel



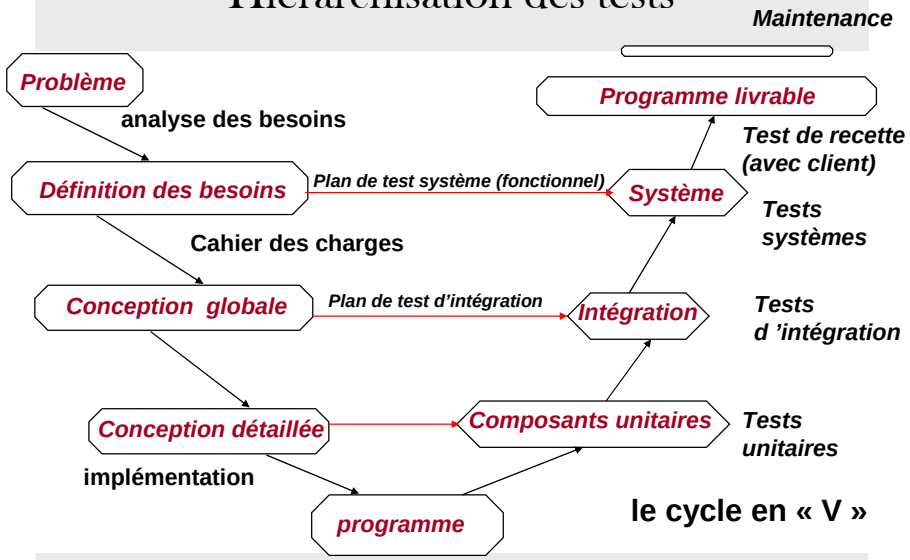
2- Rappels test de logiciel

- Le test : quoi et comment
- Etapes et processus de test
- Génération de test

Test de logiciel

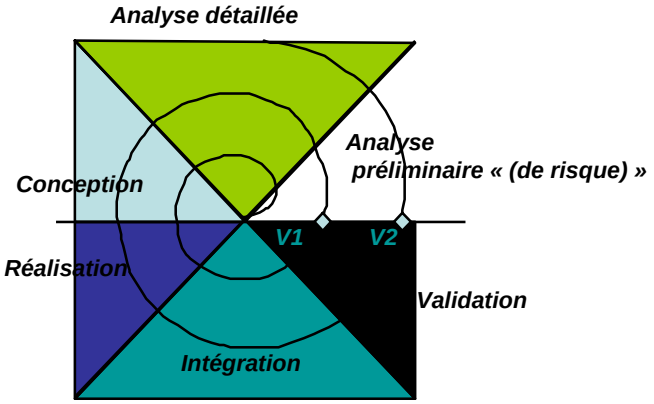
- Plusieurs échelles:
  - Unitaire, intégration, système
- Plusieurs techniques
  - Dynamique / statique
- Génération de test
  - Fonctionnel / structurel

Hiérarchisation des tests



le cycle en « V »

Cycle de vie en « spirale »



Synergie avec approche par objets

Test unitaire

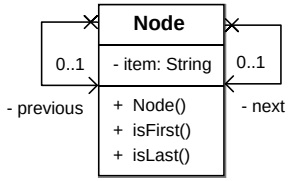
- Validation d'un module indépendamment des autres
- Valider intensivement les fonctions unitaires
- Les unités sont-elles suffisamment spécifiées?
- Le code est-il lisible, maintenable...?

Test unitaire

- Pour un langage procédural
  - unité de test = procédure

```
void Ouvrir (char *nom, Compte *C, float S, float D )
{
                    C->titulaire = AlloueEtCopieNomTitulaire(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



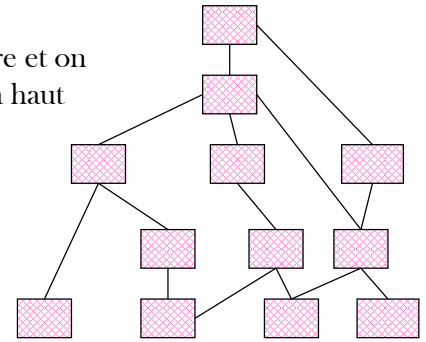
Test d'intégration

- 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

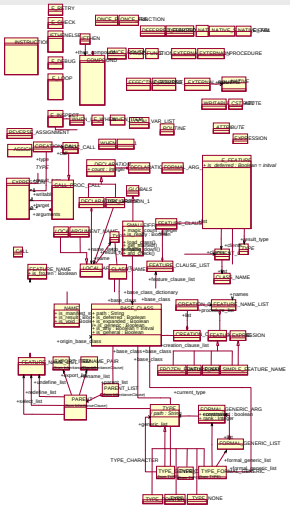


Test d'intégration

- 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



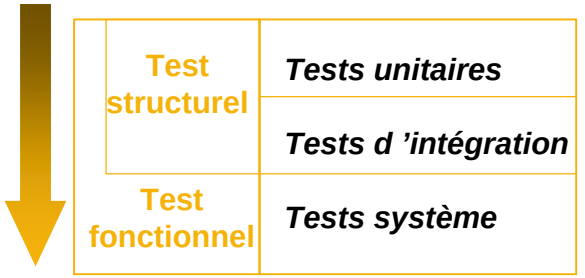
Test système

- 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

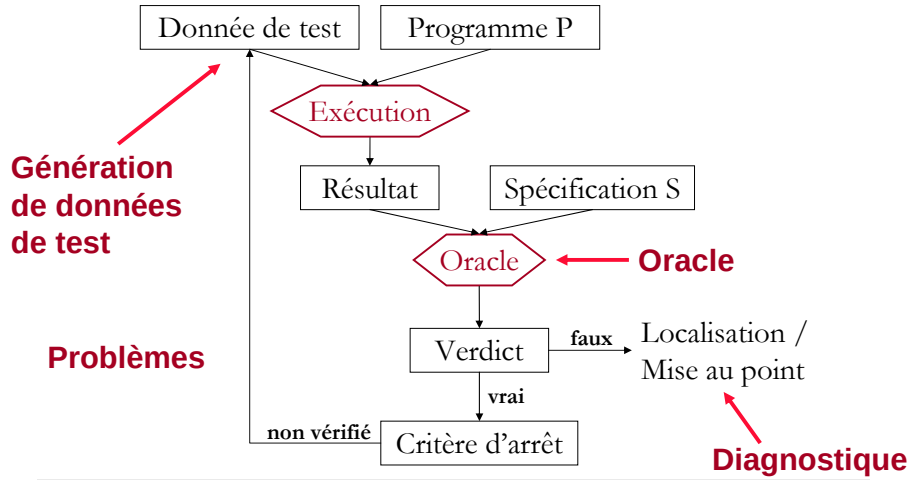
Etapes et hiérarchisation des tests



2- Rappels test de logiciel

- Le test : quoi et comment
- Etapes et processus de test
- Génération de test

Le test dynamique : processus



Le test dynamique de logiciel

- Soit D le domaine d'entrée d'un programme P spécifié par S, on voudrait pouvoir dire
  - Soit D le domaine de P: $\forall x \in D\ P(x) = S(x)$
- Test exhaustif impossible dans la plupart des cas
  - Domaine D trop grand, voire infini
  - Trop long et coûteux

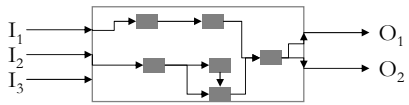
Le test dynamique

- On cherche alors un ensemble de données de test T tel que
  - $T \subset D$
  - si $\forall x \in T\ P(x) = S(x)$ alors $\forall x \in D\ P(x) = S(x)$
- Critère d'arrêt pour la génération de données de test
  - {données de test} = T

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

Génération de test

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

3- Test unitaire de composants 00

Test unitaire OO

- Tester une unité isolée du reste du système
- L'unité est la classe
  - Test unitaire = test d'une classe
- Test du point de vue client
  - les cas de tests appellent les méthodes depuis l'extérieur
  - on ne peut tester que ce qui est public
  - Le test d'une classe se fait à partir d'une classe extérieure
- Au moins un cas de test par méthode publique
- Il faut choisir un ordre pour le test
  - quelles méthodes sont interdépendantes?

Cas de test unitaire

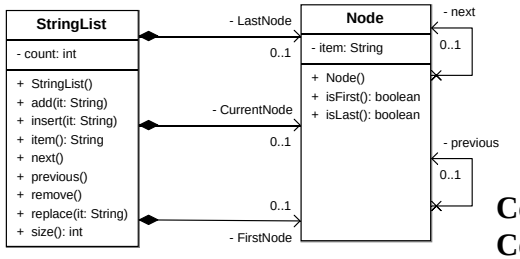
- **Cas de test = une méthode**

- Corps de la méthode
  - Configuration initiale
  - Une donnée de test
    - un ou plusieurs paramètres pour appeler la méthode testée
  - Un oracle
    - il faut construire le résultat attendu
    - ou vérifier des propriétés sur le résultat obtenu

- Une classe de test pour une classe testée
  - Regroupe les cas de test
  - Il peut y avoir plusieurs classes de test pour une classe testée

Exemple

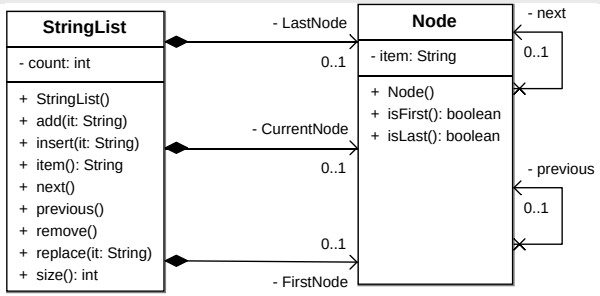
Comment tester la classe StringList?



•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?
...

Exemple : test de StringList



- Créer une classe de test qui manipule des instances de la classe StringList

- Au moins 9 cas de test (1 par méthode publique)

- Pas accès aux attributs privés : count, LastNode, CurrentNode, FirstNode

Tests: deux niveaux d'abstractions

- Dans ce cours nous optons pour deux niveaux d'abstractions pour les cas de tests

  - **Cas de Tests Abstraits** (Spécification des cas de tests au niveau modèle: utilisation des diagrammes de séquences UML)
    - **Objectifs:** ancrer les tests dès les premières étape du cycle de développement, documenter les cas de tests

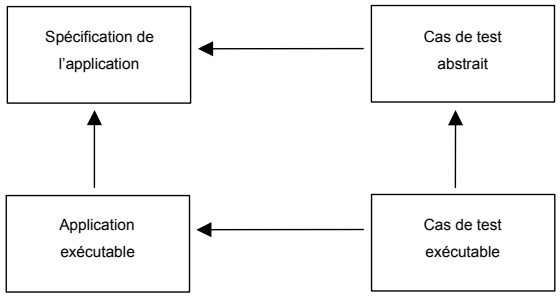
  - **Cas de Tests Exécutables** (Spécification des cas de tests au niveau du code: utilisation du Framework Java: Junit)
    - **Objectifs:** tester concrètement le code applicatif

4- Cas de Tests Abstraits

Cas de Test Abstrait

- **Un cas de test abstrait est un cas de test construit à partir de la spécification de l'application**
- Afin de réaliser et d'exécuter une suite de tests sur une application, il est nécessaire de :
  - Construire l'ensemble des cas de test abstraits composant la suite de tests
  - Construire l'ensemble des cas de test exécutables composant la suite de tests. Ces cas de test sont basés sur les cas de test abstraits et doivent pouvoir s'exécuter sur l'application
  - Construire un testeur capable d'exécuter la suite de tests sur l'application afin de rendre le verdict (comparaison entre les résultats obtenus et les résultats attendus)

Dépendances entre tests et application



Cas de Test Abstrait
Utilisation des diagrammes de Séquences UML

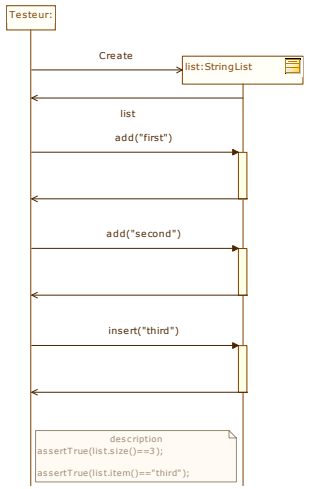
- Utilisation des diagrammes de collaboration UML afin de pouvoir spécifier des cas de test.
- **Quelques Règles :**
  - L'interaction doit obligatoirement contenir un objet représentant le testeur. Cet objet doit être identifié Testeur et ne pas avoir de type. L'objet Testeur ne doit pas être créé ni supprimé par un objet de l'interaction.
  - L'objet Testeur ne doit pas non plus recevoir de message d'appel d'opération.
  - L'interaction doit obligatoirement contenir d'autres objets. Tous les autres objets doivent être identifiés et typés. Tous ces objets doivent être créés par l'objet Testeur.
  - L'interaction peut contenir des messages d'appel d'opération synchrone ou asynchrone, mais seul l'objet Testeur peut être l'objet qui envoie ces messages.
  - L'interaction doit contenir une note contenant le résultat attendu.

Cas de Test Abstrait
Utilisation des diagrammes de Séquences UML

- Nous appellerons diagramme de séquence de test le diagramme de séquence représentant graphiquement une interaction de test.

- Ce diagramme doit respecter les contraintes suivantes :
  - L'objet Testeur doit être l'objet le plus à gauche du diagramme.
  - La note contenant le résultat attendu doit apparaître sur le diagramme, de préférence en bas, après le dernier message.

Cas de Test Abstrait: Exemple
test de StringList- la méthode insert()



5- Cas de Tests Exécutables

JUnit

- Origine
  - Xtreme programming (test-first development)
  - framework de test écrit en Java par E. Gamma et K. Beck
  - open source: www.junit.org

- Objectifs
  - test d'applications en Java
  - faciliter la création des tests
  - tests de non régression

Junit:Framework

- Le source d'un framework est disponible
- Ne s'utilise pas directement: il se spécialise

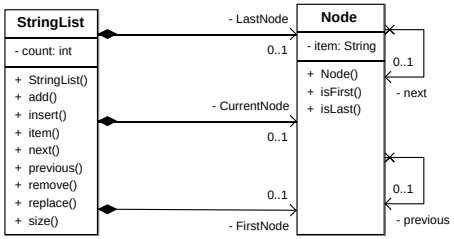
  Ex: pour créer un cas de test on hérite de la classe TestCase

  Un framework peut être vu comme un programme à « trous » qui offre la partie commune des traitements et chaque utilisateur le spécialise pour son cas particulier.

Cas de Tests Exécutables

- Dérivés des cas de tests abstraits
- **Quelques Règles:**
  - À toute interaction doit correspondre un cas de test JUnit.
    - Une classe Java qui hérite de la classe JUnit TestCase.
    - La classe doit contenir une méthode correspondant au test. Cette méthode aura pour nom testNomMethodeATester().
    - Dans la méthode testNomMethodeATester(), il doit correspondre un appel de méthode Java pour chaque message de l'interaction partant de l'objet Testeur.
    - Dans la méthode testExecutable de la classe correspondant au cas de test, il doit correspondre une assertion JUnit correspondant au résultat attendu spécifié dans l'interaction.

Cas de Tests Exécutables
test de StringList- la méthode insert()



Intention

spécification du cas de test

```
//first test for insert: call insert and see if
//current element is the one that's been inserted
public void testInsert1(){
```

initialisation

```
  StringList list = new StringList();
  list.add("first");
  list.add("second");
```

appel avec donnée de test

```
  list.insert("third");
```

oracle

```
  assertTrue(list.size()==3);
  assertTrue(list.item()=="third");
```


Conclusion

- **Les Tests: une discipline à part entière**
- Très importants dans le cycle de développement
- D'autres tests (non abordés dans ce cours)
  - Tests d'intégration, Fonctionnels, non-régression
- Les tests unitaires:
  - **Malheureusement ne couvrent pas tous les cas**
  - Restent néanmoins considérés comme un gage de qualité
  - Même si vous couvrez 99% des cas, le 1% restant peut cacher un bug majeur!!!

Lectures

- Software Engineering,
  - Ian Sommerville, Addison Wesley; 8 edition (15 Jun 2006), ISBN-10: 0321313798
- The Mythical Man-Month
  - Frederick P. Brooks JR., Addison-Wesley, 1995
- Cours de Software Engineering du Prof. Bertrand Meyer à cette @:
  - http://se.ethz.ch/teaching/ss2007/252-0204-00/lecture.html
- Cours d'Antoine Beugnard à cette @:
  - http://public.enst-bretagne.fr/~beugnard/

- UML Distilled 3rd édition, a brief guide to the standard object modeling language
  - Martin Fowler, Addison-Wesley Object Technology Series, 2003, ISBN-10: 0321193687
- UML2 pour les développeurs, cours avec exercices et corrigés
  - Xavier Blanc, Isabelle Mounier et Cédric Besse, Edition Eyrolles, 2006, ISBN-2-212-12029-X
- UML 2 par la pratique, études de cas et exercices corrigés,
  - Pascal Roques, 6ème édition, Edition Eyrolles, 2008
- Cours très intéressant du Prof. Jean-Marc Jézéquel à cette @:
  - http://www.irisa.fr/prive/jezequel/enseignement/PolyUML/poly.pdf
- La page de l'OMG dédiée à UML: http://www.uml.org/

- Design patterns. Catalogue des modèles de conception réutilisables
  - Richard Helm (Auteur), Ralph Johnson (Auteur), John Vlissides (Auteur), Eric Gamma (Auteur), Vuibert informatique (5 juillet 1999), ISBN-10: 2711786447
- **Cours sur les tests est basé sur les Cours très complets et bien faits de Yves le Traon et Benoit Baudry**