



Introduction

Tests & Validation - Séance 1



Organisation du module

- **5 séances de 4h, avec chaque séance :**
 - Présentation théorique des concepts
 - Application pratique sous forme de TP
- **2 séances de 4h, préparation du projet**
- **1 séance de 4h, évaluation finale**

Principaux concepts abordés au cours du module

Comment mesurer la qualité d'un logiciel?

Quels sont les moyens pour augmenter la qualité?

Comment gérer la vérification de la qualité pour un résultat optimum?



Comment définit-on la qualité?

Selon Larousse : **“Ensemble des caractères, des propriétés qui font que quelque chose correspond bien ou mal à sa nature, à ce qu'on en attend”**

Selon l'AFNOR (Association française de normalisation) : **“Un produit ou service de qualité est un produit dont les caractéristiques lui permettent de satisfaire les besoins exprimés ou implicites des consommateurs.”**



Comment définit-on la qualité?

Dans le domaine de l'ingénierie logicielle, il n'y a pas de définition unique, car selon le point de vue duquel on se place, on a des attentes différentes.

On peut toutefois distinguer différents paramètres et axes de la qualité logicielle.



Quels sont les axes de la qualité d'un produit?


1. L'adéquation entre le besoin et la façon dont le logiciel répond à celui-ci
2. L'absence de dysfonctionnement
3. La performance
4. L'usage naturel
5. La transparence de fonctionnement



Comment mesure-t-on la qualité?

Pour aborder la notion de qualité, comme beaucoup de notions abstraites positives, on peut la définir par l'absence du concept opposé.

De même qu'on définit la paix comme l'absence de guerre, on peut définir la qualité par l'absence de défauts.



Qu'est-ce qu'un défaut?

Un défaut est toute caractéristique qui n'est pas présente de façon souhaitée ou optimale.

Il peut se trouver dans tout aspect d'un logiciel. Y compris les aspects esthétiques.



Pourquoi mesurer la qualité?


"You can't control what you can't measure."

(Tom DeMarco)



Pourquoi mesurer la qualité?

- Un code mal maîtrisé entraîne mécaniquement une augmentation régulière et constante de la **dette technique**
- Surveillance des métriques permet une diminution du **coût de maintenance** à moyen et long terme
- Accélération de la **prise en main du code** : un code homogène sera mieux pris en charge par l'équipe, et l'intégration de nouveaux arrivants s'en retrouve également facilitée, on diminue l'**effet boîte noire**.
- **Refactoring** plus aisé.
- Qui dit approche quantitative dit **processus automatisable** et donc peu coûteux !



Qu'est-ce qu'un défaut?

Un défaut est toute caractéristique qui n'est pas présente de façon souhaitée ou optimale.

Il peut se trouver dans tout aspect d'un logiciel. Y compris les aspects esthétiques.

Quels sont les principaux axes de qualité d'un logiciel?

Le Consortium for IT Software Quality (CISQ) organise l'analyse de la qualité logicielle autour de **5 axes**:

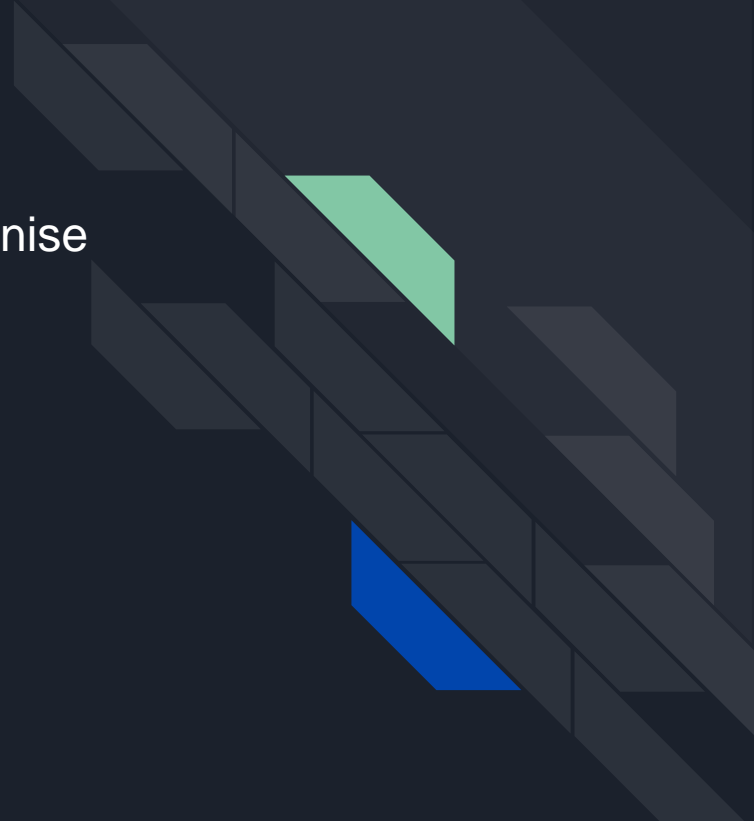
1. Fiabilité

4. Sécurité

2. Efficacité

5. Taille

3. Maintenabilité





Fiabilité

- Un attribut de résilience et de solidité structurelle.
- Elle mesure le niveau de risque d'échec de l'application.
- Le but de l'augmenter est de réduire les temps où elle n'est pas accessible ainsi que les défauts qui affectent les utilisateurs.
- Elle améliore aussi l'image du département IT au sein de l'entreprise.



Efficacité

- Le code source et l'architecture sont ce qui garantit l'efficacité du logiciel en phase d'exploitation.
- Spécialement important dans les contextes où la rapidité est importante.
- Une analyse de l'efficacité et de la scalabilité du code fournit une image claire des risques business liés aux dégradations des temps de réponse.



Sécurité

- Une mesure de la probabilité de faille due à des pratiques de code ou d'architecture
- Quantifie les vulnérabilités critiques qui sont dommageables au métier. (Perte d'informations stratégiques, fuite de données à caractère personnel, etc.)



Maintenabilité

- Inclut la notion d'adaptabilité, de portabilité et de transférabilité (d'une équipe de dev à une autre par exemple)
- C'est un aspect essentiel pour les applications critiques pour le métier. En particulier quand le « time to market » est un aspect important de la compétitivité de l'organisation.
- Il est également très important de garder sous contrôle les coûts de maintenance.

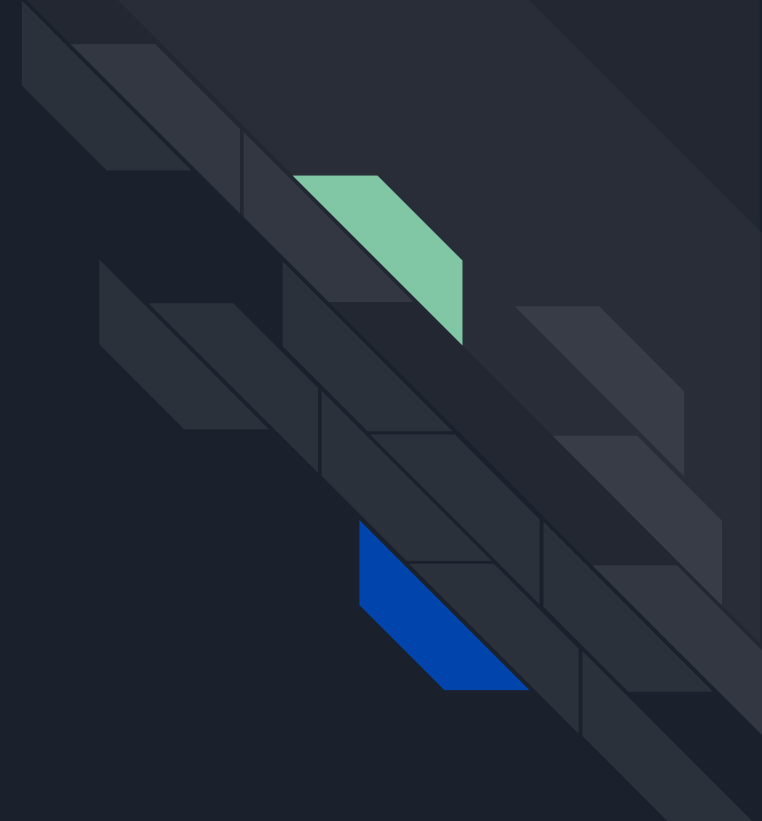


Taille

- Bien que ce ne soit pas un attribut à part entière, cette caractéristique impacte tous les autres aspects.
- Combinée avec les autres caractéristiques de qualité, la taille peut être utilisée pour estimer le volume de travail effectué ou à effectuer par les équipes.
- Elle permet également de mesurer ou d'estimer la productivité.

Principales caractéristiques souhaitées pour un logiciel

1. Documentation
2. Pratiques de code
3. Complexité
4. Standards d'Architecture
5. Taille





Documentation

- Documents externes (spécification, cahiers de tests, manuels utilisateur, etc.), mais pas uniquement.
- Le code est une forme de documentation. Le nommage des classes, membres et fonctions, ainsi que l'indentation sont donc importants !
- L'organisation du code en dossiers est relative à la lecture du code, on peut parfois la considérer comme de la documentation.



Documentation

Axes de qualité concernés directement :

- Maintenabilité

Axes de qualité concernés indirectement

- Tous les autres



Pratiques de code

- Application des principes de la POO
- Gestion d'erreurs. (Codes vs Exceptions)
- Protection des accès à chaque module, assainissement des paramètres d'appels, programmation défensive...



Pratiques de code

Axes de qualité concernés directement :

- Fiabilité,
- Sécurité,
- Efficacité
- Maintenabilité



Complexité

- Complexité cyclomatique, nombres de niveaux d'indentation au sein d'un bloc de code.
- Algorithmie (Big O Notation)
- Pratiques de programmation (usage des if/else, du polymorphisme ou de la composition).
- Présence de code mort, de code non testé.



Complexité

Axes de qualité concernés directement :

- Fiabilité,
- Maintenabilité

Axes de qualité concernés indirectement :

- Sécurité



Standards d'Architecture

- Architecture découplée (en couches, MVC, hexagonale, etc.)
- Gestion de la persistance (Data access)
- Ratios de couplage entre les modules
- Réutilisation des modules, généricité



Standards d'Architecture

Axes de qualité concernés directement :

- Fiabilité,
- Sécurité,
- Efficacité
- Maintenabilité



Taille

- Nombre de lignes de code
- Taille des fichiers binaires générés, des images Docker



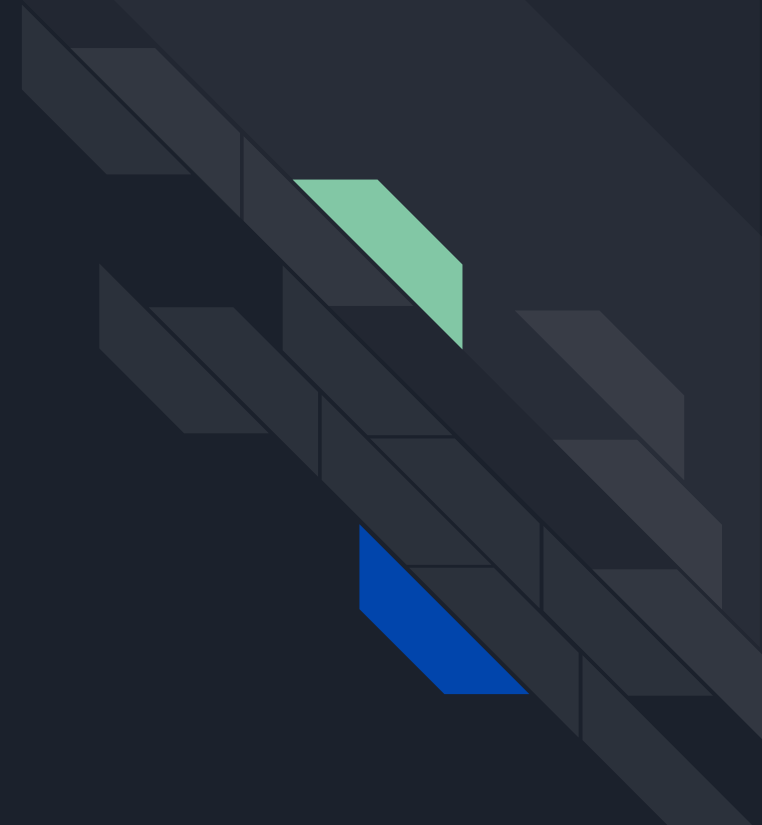
Taille

Axes de qualité concernés directement :

- Maintenabilité

Métriques logicielles

Quelques caractéristiques à mesurer pour
estimer la qualité d'un logiciel





Quelles caractéristiques peut-on mesurer pour estimer la qualité d'un logiciel?

Il y a 3 principales catégories de métriques dans la qualité logicielle:

- Maintenance applicative
- Qualité applicative
- Respect des processus de développement



Quelles caractéristiques peut-on mesurer pour estimer la qualité d'un logiciel?

D'un point de vue technique, toutes les métriques de qualité logicielle sont liées à au code et à son adéquation avec le besoin métier qu'il est censé remplir.

Par souci de simplification nous allons donc distinguer 2 catégories de caractéristiques mesurables dans un logiciel:

- Les bugs, qui représentent un comportement qui n'est pas celui qui est attendu.
- Les défauts, qui sont des caractéristiques qui rendent plus difficile l'évolution du logiciel.

Maintenabilité du logiciel



Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

(Martin Fowler)



Quelques métriques liées à la maintenabilité

- Nombre de lignes de code
- Couplage
- Complexité cyclomatique
- Cohésion
- Densité de défauts (code smells)



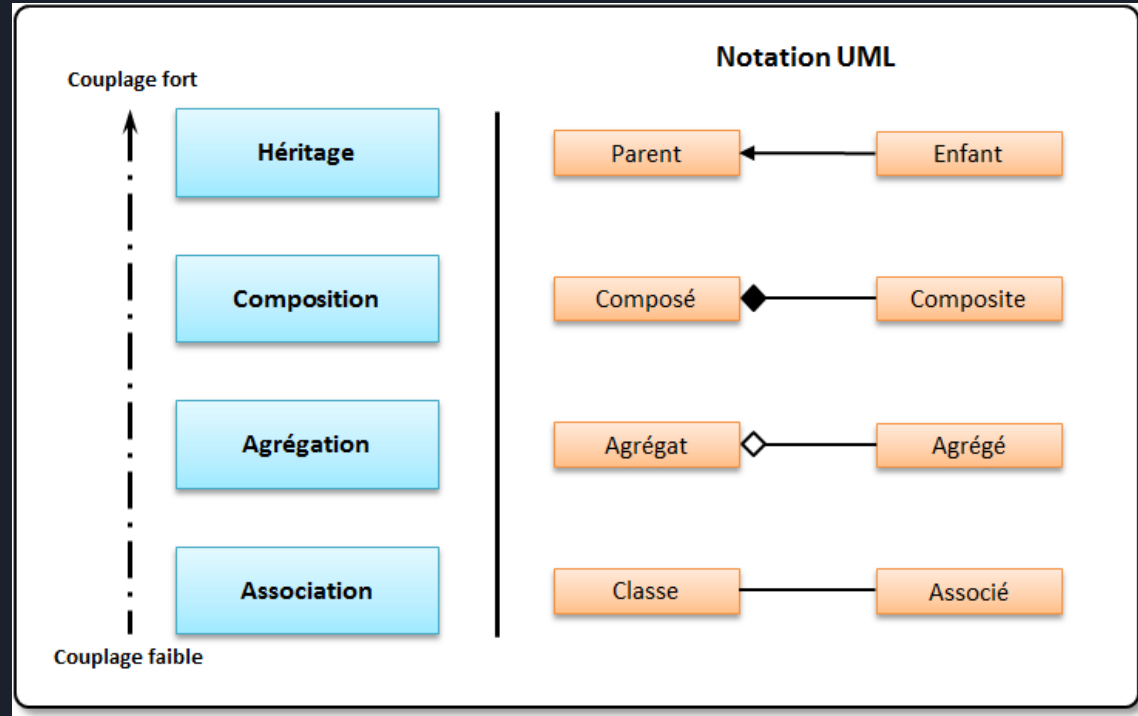
Métriques liées à la maintenabilité: Couplage

Définition :

Le couplage représente une relation entre deux éléments, classes ou modules. Il peut être « fort » ou « faible ». Son degré influe sur les répercussions des modifications d'un élément sur l'autre.

Métriques liées à la maintenabilité:

Couplage





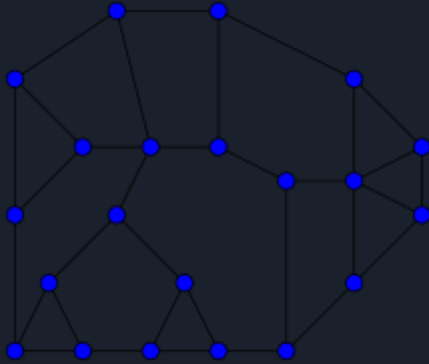
Métriques liées à la maintenabilité: Couplage

Ce qu'il faut en retenir :

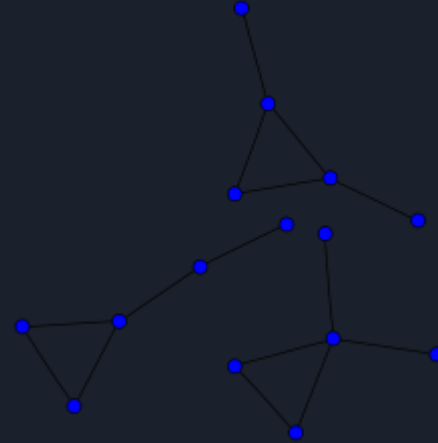
Plus on a un couplage fort entre les composants, plus le logiciel est difficile à maintenir. On favorise **toujours** un couplage **faible**.



Métriques liées à la maintenabilité: Complexité cyclomatique



Graphe connexe



Graphe non connexe
avec trois composantes
connexes



Métriques liées à la maintenabilité: Complexité cyclomatique

Définition :

$$M = E - N + 2P$$

Où

M = complexité cyclomatique ;

E = le nombre d'arêtes du graphe ;

N = le nombre de nœuds du graphe ;

P = le nombre de composantes
connexes du graphe.



Métriques liées à la maintenabilité: Complexité cyclomatique

Ce qu'il faut en retenir :

Plus on a de chemins algorithmiques, plus la complexité augmente.

Pour garantir un code de qualité, il faut diminuer les tests conditionnels.



Métriques liées à la maintenabilité: Cohésion

Définition :

Elle représente le degré d'accord entre les différents éléments d'un module. Elle peut servir à mesurer le degré d'encapsulation d'un module et le masquage de l'information.



Métriques liées à la maintenabilité: Cohésion

Selon Pressman², il existe sept niveaux de cohésion :

1. **Accidentel** : décrivant le niveau le plus faible où le lien entre les différentes méthodes est inexistant ou bien créé sur la base d'un critère futile.
2. **Logique** : lorsque les méthodes sont reliées logiquement par un ou plusieurs critères communs.
3. **Temporel** : lorsque les méthodes doivent être appelées au cours de la même période de temps.

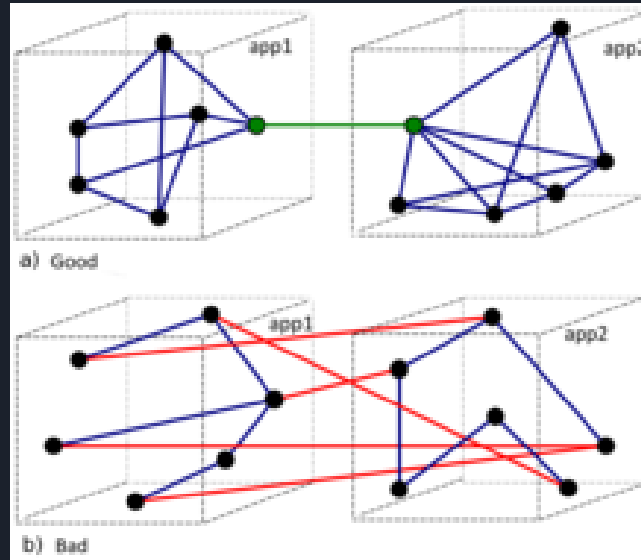


Métriques liées à la maintenabilité: Cohésion

4. **Procédural** : lorsque les méthodes doivent être appelées dans un ordre spécifique.
5. **Communicationnel** : lorsque les méthodes manipulent le même ensemble spécifique de données.
6. **Séquentiel** : lorsque les méthodes qui manipulent le même ensemble de données doivent être appelées dans un ordre spécifique.

Métriques liées à la maintenabilité: Cohésion

7. **Fonctionnel** : réalise le niveau le plus élevé lorsque la classe ou le module est dédié à une seule et unique tâche bien spécifique.






Métriques liées à la maintenabilité: Cohésion

Ce qu'il faut en retenir :

Le niveau accidentel est celui de plus **faible cohésion**, le niveau fonctionnel celui de plus **forte cohésion**.

Une bonne architecture logicielle nécessite **la plus forte cohésion possible**.



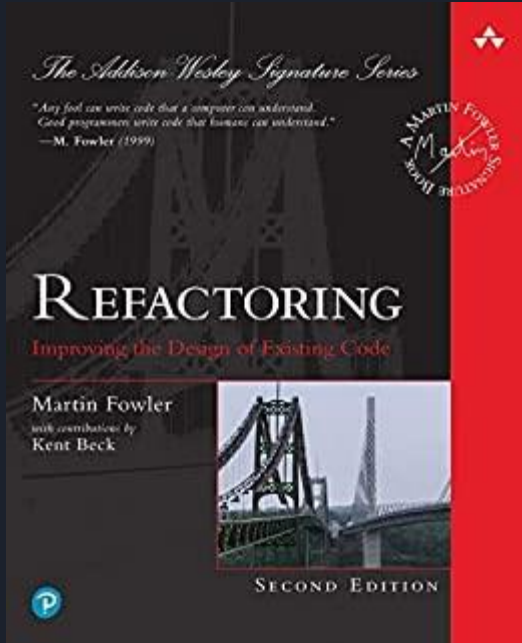
Métriques liées à la maintenabilité: Densité de défauts: les « **code smells** »

Définition :

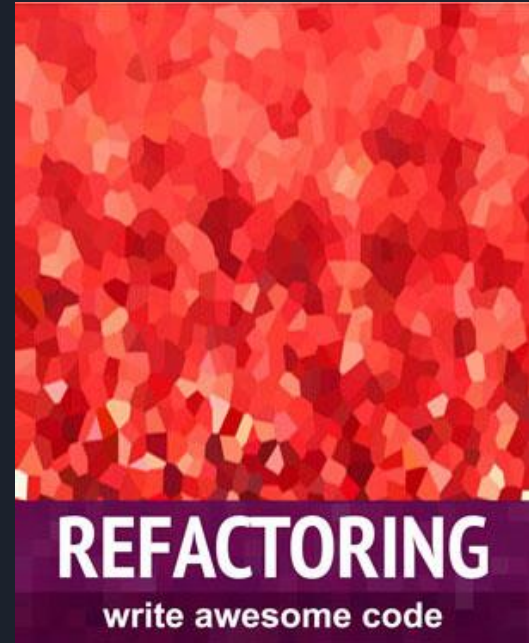
Ils représentent une caractéristique qui indique la probabilité d'un problème plus important.

Défini par Kent Beck dans les années 90, le terme a gagné en popularité après l'utilistion du concept par Martin Fowler dans son célèbre ouvrage « Refactoring ».

Métriques liées à la maintenabilité: Les « **code smells** »



Martin Fowler



[Refactoring.guru](https://refactoring.guru)



Exemples de « code smells »

Bloaters: gonflent les classes/méthodes

- **Long Method** : toute méthode faisant plus de 30 lignes de code
- **Long Class** : Toute classe contenant trop de champs, méthodes, etc...
- **Large parameter list** : Méthode prenant plus de 3 arguments



Exemples de « code smells »

Change preventers : Empêchent la modification facile

- **Large switch statement** : If/Else comprenant trop de conditions
- **Shotgun surgery** : Tout changement dans une classe doit se répercuter dans plusieurs autres classes
- **Parallel inheritance hierarchy** : Quand on crée une classe enfant, il faut créer une classe enfant pour un autre parent.



Exemples de « code smells »

Dispensables : code qu'on pourrait supprimer


- **Commentaires inutiles** : tout commentaire qui « explique » le code
- **Code mort** : Code qui n'est jamais appelé, ou commenté
- **Speculative Generality** : Code écrit « pour plus tard »



Exemples de « code smells »

Couplers : code qui génère du couplage

- **Feature Envy** : Un traitement sur une donnée est régulièrement fait en dehors de la classe qui contient la donnée
- **Message chains**: Exemple `e = a.getB().getC().getExample()`
- **Middle Man** : Classe ne faisant que déléguer un appel à une autre classe.



Merci
pour votre
attention