# Évolution et restructuration des logiciels

## Rapport

SAR Alexandre

October 2024


**Lien vers le dépôt GitHub :**
https://github.com/texao/tp1EvolutionRestructurationLogiciel

# Contents

# 1 Introduction

Ce rapport se concentre sur l'analyse du couplage entre classes au sein d'une application, en introduisant des métriques de couplage pour quantifier les relations entre les classes et en visualisant ces relations à l'aide de graphes. Nous examinerons également les avantages d'une organisation modulaire, qui peut être réalisé par le regroupement des classes en modules basés sur leurs interactions.

Nous allons définir les métriques de couplage. Nous présenterons ensuite la génération d'un graphe de couplage, ainsi que les analyses de celui-ci. La deuxième partie du rapport sera dédiée à l'identification de modules, décrivant l'objectif et la méthode de regroupement des classes, suivie d'une analyse des résultats obtenus.

Ce rapport vise à démontrer l'importance du couplage entre classes et à proposer des approches pour améliorer la structure des applications, renforçant leur qualité et leur maintenabilité.

# 2 Couplage entre classes

## 2.1 métrique de couplage

La métrique de couplage est un indicateur essentiel pour évaluer les dépendances entre les classes d'une application. Nous avons défini le couplage entre deux classes A et B :

$$Couplage(A,B) = \frac{Nombrederelations(A.mi, B.mj)}{Nombretotalderelationsbinairesentrelesméthodesdetouteslesclasses} \quad (1)$$

**Interprétation du Couplage**

Le couplage calculé entre les classes fournit des informations précieuses sur leur interdépendance. Une valeur élevée du couplage indique une forte dépendance entre les classes. Un couplage faible suggère une plus grande modularité et une indépendance accrue des classes, favorisant ainsi la maintenance et la réutilisation du code.

Pour illustrer cette métrique, j'ai calculé le couplage entre plusieurs paires de classes dans notre application. Ces calculs ont permis de mieux comprendre les relations existantes entre les différentes classes et d'identifier celles qui sont fortement couplées.

Plusieurs calculs de couplage ont été effectués. Voici un exemple de couplage pour la classe Order avec d'autres classes.

```
Couplages de la classe Order:
  Avec Order: 0,1600
  Avec Shop: 0,0000
  Avec Product: 0,1200
  Avec Main: 0,0000
  Avec Inventory: 0,0800
```

Figure 1: Couplage entre classes

Dans la figure 1, nous constatons que les couplages de la classe Order avec d'autres classes. Par exemple pour avec la classe Order, il y a un couplage de 0,16 entre la classe Order et elle-même. Cela peut indiquer que la classe Order appelle fréquemment ses propres méthodes, cela montre que cette classe gère ses propres opérations internes.

Nous remarquons également que avec Shop et Main il y a une valeur de 0,0000. Cette valeur indique qu'il n'y a pas de relations d'appel entre la classe Order et la classe Shop. Cela montre que ces deux classes fonctionnent de manière indépendante, sans dépendances directes entre elles.

Avec Product, il y a un couplage de 0,1200 entre Order et Product signifie qu'il y a une certaine interaction entre ces deux classes, où Order appelle certaines méthodes de Product. Cela pourrait indiquer que Order a besoin d'accéder à des informations sur Product pour créer ou gérer des commandes.

## 2.2 graphe de couplage

Le graphe de couplage est une représentation visuelle des interactions entre les différentes classes. Il illustre les dépendances entre les classes en mettant en évidence les relations d'appel entre les méthodes. Chaque nœud du graphe représente une classe, alors que les arêtes indiquent les appels de méthode entre ces classes, pondérés par une métrique de couplage.

### 2.2.1 Génération du Graphique

1. **Écriture des Relations d'Appel au Format DOT** : La méthode writeCallRelationsToDot est utilisée pour créer un fichier .dot, qui est un format de description de graphes. Elle génère le contenu nécessaire pour construire le graphe. Le fichier .dot résultant sert de base pour la visualisation des relations de couplage.

2. **Génération de l'Image PNG** : Après la création du fichier .dot, la méthode generatePNG est appelée pour convertir ce fichier en une image PNG. Cette méthode utilise l'outil dot de Graphviz, qui permet de transformer la représentation textuelle du graphe en une image visuelle. La génération de l'image facilite l'analyse visuelle des relations de couplage entre les classes.

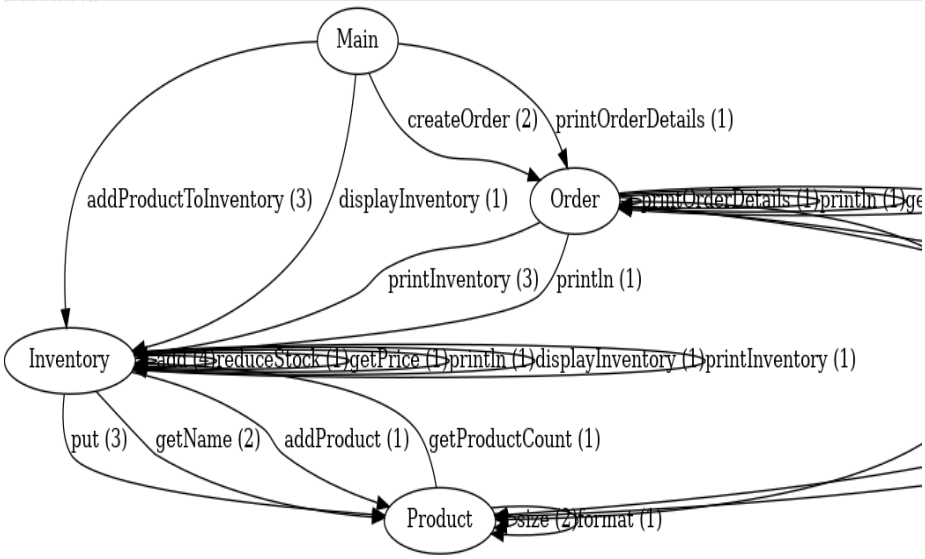


Figure 2: Graphe de couplage

### 2.2.2 Analyse du graphe

Le graphe ci-dessus illustre les relations de couplage entre les classes. Chaque nœud représente une classe, alors que chaque arête correspond à un appel de méthode entre deux classes, avec un poids indiquant le nombre d'appels pour une méthode spécifique.

Dans ce graphe, plusieurs observations peuvent être faites :

La classe `Inventory` apparaît comme centrale, elle est fortement couplée avec plusieurs autres classes, notamment `Product` et `Order`. Elle est appelée par plusieurs méthodes comme `put` avec une pondération de 3, `reduceStock`, `addProduct`, et `getPrice`, ce qui montre qu'elle joue un rôle crucial dans la gestion des stocks de produits.

La classe `Order` interagit principalement avec `Inventory` et `Product`. Les méthodes `printInventory`, `createOrder`, et `getProduct` montrent que `Order` dépend directement de ces classes pour traiter et afficher les informations de commandes et de stocks.

La classe `Main` est responsable de la création et de la gestion des objets principaux du système, avec des appels vers `Inventory` pour ajouter des produits (`addProductToInventory`) et vers `Order` pour afficher les détails des commandes (`printOrderDetails`).

La classe `Product`, moins centrale, interagit régulièrement avec `Inventory`, principalement pour les méthodes `getProductCount` et `addProduct`, ce qui souligne son rôle dans la gestion des informations sur les produits disponibles.

# 3 Identification de Modules

## 3.1 Objectif

L'objectif est de définir un algorithme de regroupement hiérarchique (clustering) permettant d'identifier des groupes de classes fortement couplées dans une application. Ces groupes de classes peuvent être interprétés comme des services, composants, ou modules fonctionnels.

### 3.1.1 Méthode de regroupement

J'ai créé une classe ModuleIdentification. Elle permet d'identifier et regrouper des classes dans des modules en fonction de leurs relations de couplage. Le processus d'identification des modules s'effectue en plusieurs étapes :

1. **Initialisation des Clusters** : Chaque classe est initialement considérée comme un cluster distinct. Cela permet de commencer le processus de regroupement à partir d'un état où chaque classe n'est pas regroupé.
2. **Fusion des Clusters** : L'algorithme fusionne les clusters en se basant sur les paires de classes ayant le couplage maximal. Cette étape est répétée jusqu'à ce qu'un nombre défini de modules soit atteint ou que plus aucune fusion ne soit possible. Les regroupements réalisés sont enregistrés pour pouvoir par la suite faire une analyse.
3. **Filtrage des Modules** : Une fois les clusters fusionnés, une étape de filtrage est appliquée pour conserver uniquement les modules répondant à des critères spécifiques, comme un couplage moyen minimal et un nombre maximal de modules.
4. **Calculs de Couplage** : Les calculs de couplage entre les classes sont effectués, ce qui permet d'évaluer les relations entre les classes au sein de chaque cluster ainsi qu'entre différents clusters.

Cette approche permet de structurer l'application de manière modulaire. En regroupant les classes en fonction de leur couplage, le système peut être étendu ou modifié sans l'application.

## 3.2 Regroupement des classes

### 3.2.1 Résultats du regroupement

```
Regroupement: [Inventory] dans [Product, Inventory]
Regroupement: [Order] dans [Product, Inventory, Order]
Regroupement: [Main] dans [Product, Inventory, Order, Main]
Module: [Product, Inventory, Order, Main], Average Coupling: 0.7250000000000001
```

Ci dessus, nous retrouvons les résultats du regroupement des classes.

### 3.2.2 Analyse

Voici les résultats des regroupements obtenus lors de l'exécution de l'algorithme:

- La classe `Inventory` a été fusionnée avec le cluster contenant les classes `Product` et `Inventory`, formant ainsi un nouveau cluster.
- La classe `Order` a ensuite été ajoutée au cluster précédent, composé de `Product` et `Inventory`.
- Enfin, la classe `Main` a été regroupée avec le cluster formé de `Product`, `Inventory`, et `Order`, formant ainsi un module complet.

À la fin du processus de regroupement, un module a été identifié, contenant les classes `Product`, `Inventory`, `Order`, et `Main`. Ce module a un couplage moyen de `0.725`, il s'agit d'un un niveau élevé de dépendances entre ces classes.

Le processus de regroupement a permis de structurer l'application en modules basés sur les relations de couplage, ce qui favorise une meilleure organisation modulaire. Cette approche présente plusieurs avantages :

- **Facilité de Maintenance** : En regroupant les classes fortement couplées en modules cohérents, il devient plus facile de gérer et de maintenir le code. Les modifications apportées à un module n'affectent généralement pas d'autres modules.

- **Amélioration de la Réutilisabilité** : Les modules bien définis peuvent être réutilisés dans d'autres parties de l'application ou même dans d'autres projets. Cette réutilisabilité favorise l'efficacité du développement.

# 4 Conclusion

Nous avons exploré le concept de couplage entre classes, en mettant en évidence son rôle dans l'analyse de la structure d'une application logicielle. Grâce à la définition de métriques de couplage, nous avons pu quantifier les dépendances entre les classes et mieux comprendre leurs interactions. La génération d'un graphe de couplage a permis de visualiser ces relations, facilitant l'identification des zones de forte dépendance.

Nous avons également présenté une méthode d'identification de modules, qui a permis de regrouper les classes en clusters basés sur leur couplage. Ce processus a conduit à une meilleure organisation modulaire.

En conclusion, l'analyse du couplage entre classes et l'identification de modules sont des étapes cruciales pour améliorer la qualité du code et la maintenabilité des applications. Les résultats obtenus montrent l'importance d'adopter des pratiques de conception orientées vers une architecture modulaire.