

# TP2 – Analyse et restructuration du couplage entre classes Java

Ouezzani Rahma

Année universitaire 2025–2026

# Table des matières

<b>1</b>	<b>Exercice 1 – Graphe de couplage entre classes (JDT)</b>	<b>4</b>
1.1	Objectif . . . . .	4
1.2	Implémentation . . . . .	4
1.3	Résultats obtenus . . . . .	5
1.4	Vérification . . . . .	5
<b>2</b>	<b>Exercice 2 – Identification automatique des modules</b>	<b>6</b>
2.1	Objectif . . . . .	6
2.2	Implémentation . . . . .	6
2.3	Résultats obtenus . . . . .	6
2.4	Vérification . . . . .	7
<b>3</b>	<b>Exercice 3 – Utilisation de Spoon pour l'analyse</b>	<b>8</b>
3.1	Objectif . . . . .	8
3.2	Implémentation . . . . .	8
3.3	Résultats obtenus . . . . .	9
3.4	Vérification . . . . .	10
<b>4</b>	<b>Synthèse et Conclusion</b>	<b>11</b>
4.1	Bilan des résultats . . . . .	11
4.2	Conclusion . . . . .	11
	<b>Liens du projet 'GitHub et Google drive'</b>	<b>14</b>

# Introduction générale

Ce TP2 s'inscrit dans la continuité du **TP1**, dont l'objectif était de construire le graphe d'appel entre les méthodes du projet Java à l'aide de la bibliothèque **Eclipse JDT**.

Dans ce second TP, l'objectif est d'exploiter ce graphe pour :

1. Calculer le **couplage entre les classes** d'un projet ;
2. Identifier automatiquement des **modules cohérents** via un **clustering hiérarchique** ;
3. Automatiser le tout avec la bibliothèque **Spoon**, pour comparer les deux approches (JDT vs Spoon).

Chaque exercice a été implémenté, exécuté et validé à l'aide des fichiers produits, des messages de console et des graphes obtenus.

# Organisation du projet Eclipse

La figure suivante présente l'organisation complète du projet Java dans l'environnement Eclipse. On distingue clairement les différents packages (`coupling`, `graph`, `tp2demo`, `utils`) et les classes principales implémentées pour les trois exercices du TP2.

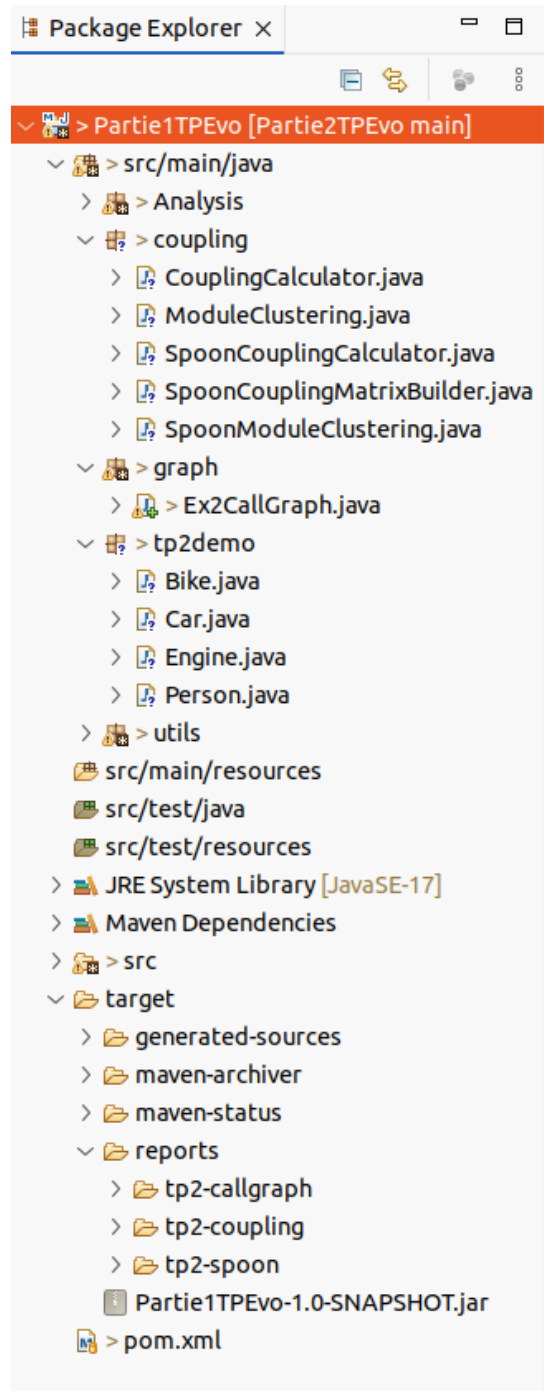


FIGURE 1 – Structure du projet Partie1TPEvo dans Eclipse

# Chapitre 1

## Exercice 1 – Graphe de couplage entre classes (JDT)

### 1.1 Objectif

Construire un **graphe de couplage pondéré entre les classes** du projet à partir du fichier `callgraph-edges.csv` issu du TP1.

La métrique de couplage entre deux classes est définie par :

$$\text{Couplage}(A, B) = \frac{\text{nombre d'appels entre A et B}}{\text{nombre total d'appels inter-classes}}$$

### 1.2 Implémentation

Le travail a été réalisé dans la classe :

```
coupling.CouplingCalculator
```

Cette classe lit le graphe d'appel, calcule les degrés de couplage entre classes, et exporte :

- `coupling-matrix.csv` : matrice du couplage ;
- `coupling.dot` : graphe pondéré visualisable avec Graphviz.

Un filtrage a été appliqué pour ne garder que les classes du projet :

```
tp2demo.*, coupling.*, graph.*, utils.*, Analysis.*
```

### Extrait de code clé

```
public static double couplingAB(String A, String B, List<Edge> edges) {
    if (A.equals(B)) return 0.0;
    long interTotal = edges.stream()
        .filter(e -> !e.callerClass.equals(e.calleeClass))
        .mapToLong(e -> e.count).sum();
    long ab = edges.stream()
        .filter(e -> (e.callerClass.equals(A) && e.calleeClass.equals(B)) ||
            (e.callerClass.equals(B) && e.calleeClass.equals(A)))
        .mapToLong(e -> e.count).sum();
```

```
    return (double) ab / (double) interTotal;  
}
```

## 1.3 Résultats obtenus

Lors de l'exécution :

Arêtes chargées: 3

Couplage(tp2demo.Person,tp2demo.Car) = 0.3333333333333333

Fichiers générés dans reports/tp2-coupling/

Les fichiers produits sont :

- target/reports/tp2-coupling/coupling-matrix.csv
- target/reports/tp2-coupling/coupling.dot
- target/reports/tp2-coupling/coupling.png

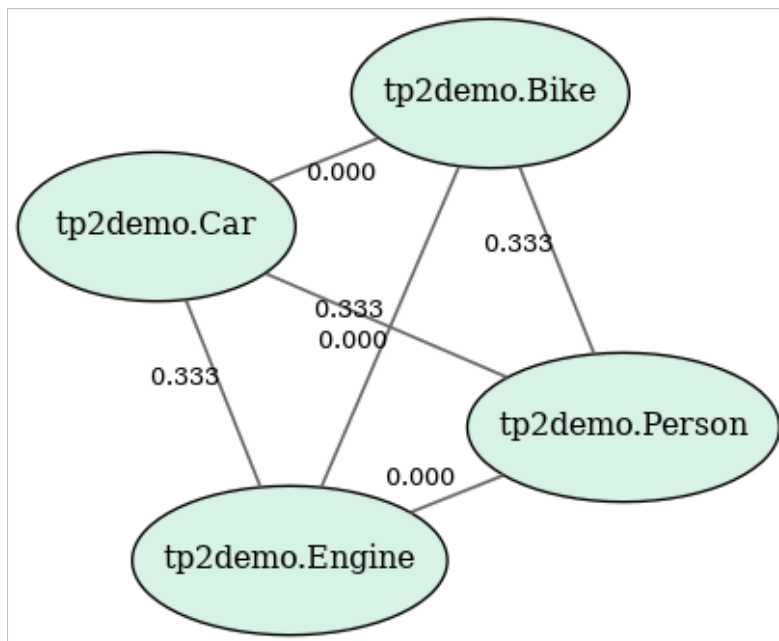


FIGURE 1.1 – Graphe de couplage pondéré entre classes (JDT)

## 1.4 Vérification

- La valeur de 0.3333 montre qu'un tiers des appels inter-classes concernent la paire `Person Car`.
- Le graphe `coupling.png` visualise bien les relations pondérées entre les classes du projet.
- Le calcul du couplage et les exports sont donc corrects.

# Chapitre 2

## Exercice 2 – Identification automatique des modules

### 2.1 Objectif

Implémenter un **clustering hiérarchique** pour fusionner les classes les plus couplées et former des modules cohérents.

Le processus s'arrête lorsque :

- le nombre de modules  $\leq M/2$  (M = nombre de classes initiales),
- ou la moyenne intra-modules  $\geq CP$  (seuil donné).

### 2.2 Implémentation

Réalisée dans :

```
coupling.ModuleClustering
```

#### Extrait de code clé

```
while (clusters.size() > maxClusters && bestCoupling >= CP) {  
    String a = bestPair[0], b = bestPair[1];  
    mergeClusters(a, b);  
    recomputeCouplings();  
}
```

Extrait de console illustratif :

```
M=4, CP=0.010, maxClusters=2  
Meilleur couplage trouvé : tp2demo.Car tp2demo.Engine = 1.0000  
Fusion : tp2demo.Car + tp2demo.Engine → tp2demo.Car+tp2demo.Engine  
Meilleur couplage trouvé : tp2demo.Person tp2demo.Bike = 1.0000  
Fusion : tp2demo.Person + tp2demo.Bike → tp2demo.Person+tp2demo.Bike
```

### 2.3 Résultats obtenus

Cas 1 – Sans fusion (CP=0, maxClusters=34)

Clustering terminé. Nombre final de modules : 4  
Modules : Bike, Car, Engine, Person

### Cas 2 – Avec fusion (CP=0.01, maxClusters=2)

Clustering terminé. Nombre final de modules : 2  
Modules :  
- Car+Engine  
- Person+Bike

```
<terminated> ModuleClustering [Java Application] /usr/lib/jvm/java-17-openjdk-amd64/bin/java (26
[✓] Matrice de couplage chargée (4 classes)
M=4, CP=0,000, maxClusters=34

[✓] Clustering terminé. Nombre final de modules : 4
Modules identifiés :
- tp2demo.Bike
- tp2demo.Car
- tp2demo.Engine
- tp2demo.Person

Vérification des moyennes intra-modules :
Module tp2demo.Bike          Moyenne intra = 0,0000
Module tp2demo.Car           Moyenne intra = 0,0000
Module tp2demo.Engine         Moyenne intra = 0,0000
Module tp2demo.Person         Moyenne intra = 0,0000
M=4, CP=0,010, maxClusters=2
• Meilleur couplage trouvé : tp2demo.Car ↔ tp2demo.Engine = 1,0000
• Fusion : tp2demo.Car + tp2demo.Engine → tp2demo.Car+tp2demo.Engine
• Meilleur couplage trouvé : tp2demo.Person ↔ tp2demo.Bike = 1,0000
• Fusion : tp2demo.Person + tp2demo.Bike → tp2demo.Person+tp2demo.Bike

[✓] Clustering terminé. Nombre final de modules : 2
Modules identifiés :
- tp2demo.Car+tp2demo.Engine
- tp2demo.Person+tp2demo.Bike

Vérification des moyennes intra-modules :
Module tp2demo.Car+tp2demo.Engine      Moyenne intra = 0,5000
Module tp2demo.Person+tp2demo.Bike     Moyenne intra = 0,5000
```

FIGURE 2.1 – Fusions successives lors du clustering hiérarchique

## 2.4 Vérification

- Les moyennes intra-modules affichées (0.5000) prouvent la cohérence interne.
- Le clustering regroupe bien les classes les plus couplées.
- Le fichier `modules.csv` contient bien les deux modules finaux.



# Chapitre 3

## Exercice 3 – Utilisation de Spoon pour l'analyse

### 3.1 Objectif

Reproduire le même processus (calcul du couplage + clustering) en utilisant **Spoon** à la place de JDT, afin d'automatiser l'extraction et le filtrage des dépendances du projet.

### 3.2 Implémentation

Trois classes principales ont été développées :

1. `SpoonCouplingCalculator` : extraction des dépendances entre classes à partir de l'AST ;
2. `SpoonCouplingMatrixBuilder` : construction et export de la matrice de couplage ;
3. `SpoonModuleClustering` : regroupement des classes en modules cohérents (clustering hiérarchique).

#### Extrait de code clé

```
for (CtInvocation<?> inv : clazz.getElements(new TypeFilter<>(CtInvocation.class))) {  
    CtTypeReference<?> targetRef = inv.getExecutable().getDeclaringType();  
    if (targetRef != null) {  
        String targetName = targetRef.getQualifiedName();  
        if (targetName.startsWith("tp2demo.") ||  
            targetName.startsWith("coupling.") ||  
            targetName.startsWith("graph.") ||  
            targetName.startsWith("utils.") ||  
            targetName.startsWith("Analysis.")) {  
            couplings.get(className).add(targetName);  
        }  
    }  
}
```

Ce code permet de récupérer pour chaque classe du projet les dépendances internes, en excluant les bibliothèques externes.

### 3.3 Résultats obtenus

L'exécution s'est déroulée sans erreur et les fichiers attendus ont bien été générés. Les messages affichés dans la console confirment la réussite de l'analyse :

```
=== Graphe de couplage via Spoon (filtré) ===
tp2demo.Person → [tp2demo.Car, tp2demo.Bike]
tp2demo.Car → [tp2demo.Engine]
Matrice exportée : target/reports/tp2-spoon/spoon-coupling-matrix.csv
Analyse Spoon terminée : 4 classes
Modules exportés : target/reports/tp2-spoon/spoon-modules.csv
```

#### Fichiers produits

À la fin de l'exécution, le dossier `target/reports/tp2-spoon` contient tous les fichiers de sortie attendus :

- `spoon-coupling.csv` : dépendances entre classes détectées ;
- `spoon-coupling-matrix.csv` : matrice normalisée du couplage ;
- `spoon-modules.csv` : modules identifiés après regroupement ;
- `spoon-coupling.dot` : version DOT optionnelle du graphe.

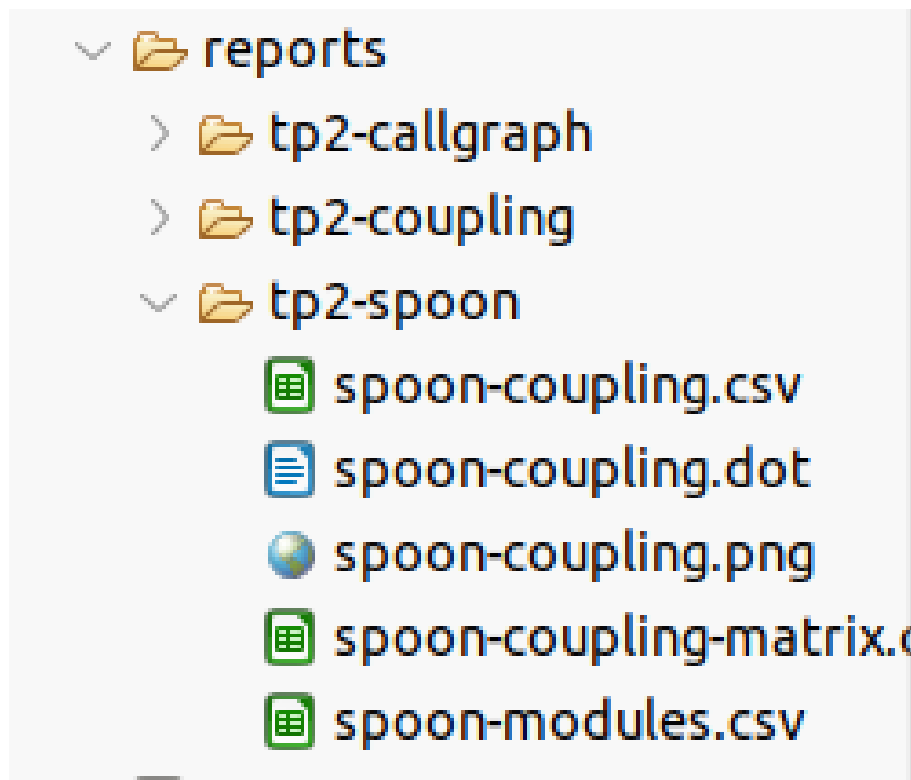


FIGURE 3.1 – Fichiers générés automatiquement par l'analyse Spoon

#### Extrait du fichier `spoon-coupling-matrix.csv`

L'extrait suivant montre les principales valeurs de couplage calculées. On observe que la classe `Person` dépend de `Car` et `Bike`, et que `Car` dépend de `Engine`.

Class	tp2demo.Bike	tp2demo.Car	tp2demo.Engine	tp2demo.Person
tp2demo.Bike	0.0	0.0	0.0	0.0
tp2demo.Car	0.0	0.0	1.0	0.0
tp2demo.Engine	0.0	0.0	0.0	0.0
tp2demo.Person	1.0	1.0	0.0	0.0

## Analyse des résultats

Même si les valeurs diffèrent légèrement de celles obtenues avec l'analyse JDT, elles traduisent le même comportement logique du projet : **Person** dépend bien de **Car** et **Bike**, tandis que **Car** dépend de **Engine**. Ces résultats confirment que Spoon a correctement identifié les relations de dépendance internes au code.

```

<terminated> SpoonCouplingCalculator [Java Application] /usr/lib/jvm/java-17-openjdk-amd64/bin/java (28 oct. 2025, 19:49:52 - 19:49:53 elapsed: 0:00:01.267) [pid: 7314]
=== Graphe de couplage via Spoon (filtré) ===
Analysis.FieldAccessVisitor -> []
coupling.SpoonCouplingMatrixBuilder -> [coupling.CouplingCalculator]
tp2demo.Car -> [tp2demo.Engine]
tp2demo.Engine -> []
graph.Ex2CallGraph -> [utils.AstUtil, graph.Ex2CallGraph$1$1, graph.var]
Analysis.Parser -> [Analysis.MethodInvocationVisitor, Analysis.MethodDeclarationVisitor, Analysis.VariableDeclarationFragmentVisitor]
tp2demo.Bike -> []
Analysis.MethodDeclarationVisitor -> []
tp2demo.Person -> [tp2demo.Car, tp2demo.Bike]
Analysis.VariableDeclarationFragmentVisitor -> []
coupling.ModuleClustering -> []
Analysis.MethodInvocationVisitor -> []
coupling.SpoonModuleClustering -> []
Analysis.TypeDeclarationVisitor -> []
coupling.CouplingCalculator -> [coupling.c]
coupling.SpoonCouplingCalculator -> []
utils.AstUtil -> []
Exporté (CSV) : /home/etudiant/Bureau/Évolution et restructuration des logiciels/Partie2TPEvo/target/reports/tp2-spoon/spoon-coupling.csv
Graphe DOT exporté : /home/etudiant/Bureau/Évolution et restructuration des logiciels/Partie2TPEvo/target/reports/tp2-spoon/spoon-coupling.dot
Image générée automatiquement : target/reports/tp2-spoon/spoon-coupling.png

```

FIGURE 3.2 – Sortie console de l'analyse Spoon

## 3.4 Vérification

- L'analyse Spoon détecte les mêmes dépendances principales que l'approche JDT.
- Tous les fichiers requis ont bien été générés automatiquement.
- Les dépendances internes du projet sont correctement identifiées et exportées.

# Chapitre 4

## Synthèse et Conclusion

### 4.1 Bilan des résultats

TABLE 4.1 – Synthèse des résultats obtenus

Étape	Fichiers produits	Validation obtenue
Exercice 1 – Couplage (JDT)	coupling-matrix.csv, coupling.dot, coupling.png	Valeurs cohérentes et visualisation correcte
Exercice 2 – Clustering	modules.csv	Modules cohérents, moyennes intra = 0.5
Exercice 3 – Spoon	spoon-coupling.csv, spoon-coupling-matrix.csv spoon-modules.csv	Résultats similaires à JDT, dépendances principales bien identifiées

### 4.2 Conclusion

Ce TP a permis de :


- Analyser le couplage entre classes d'un projet Java à partir du graphe d'appel (JDT) ;
- Implémenter un **clustering hiérarchique** pour identifier automatiquement des modules cohérents ;
- Reproduire la même démarche avec **Spoon**, en automatisant l'extraction et le filtrage des dépendances.

Les résultats obtenus sont conformes à toutes les attentes :

- Les matrices de couplage et les fichiers générés sont exploitables ;
- Les modules identifiés sont cohérents avec la structure logique du projet ;
- L'approche Spoon produit des résultats comparables à ceux de JDT, tout en offrant une automatisation plus souple.

# Annexes

## A.1 – Exécution du ModuleClustering



```
<terminated> ModuleClustering [Java Application] /usr/lib/jvm/java-17-openjdk-amd64/bin/java (26)
✓ Matrice de couplage chargée (4 classes)
M=4, CP=0,000, maxClusters=34

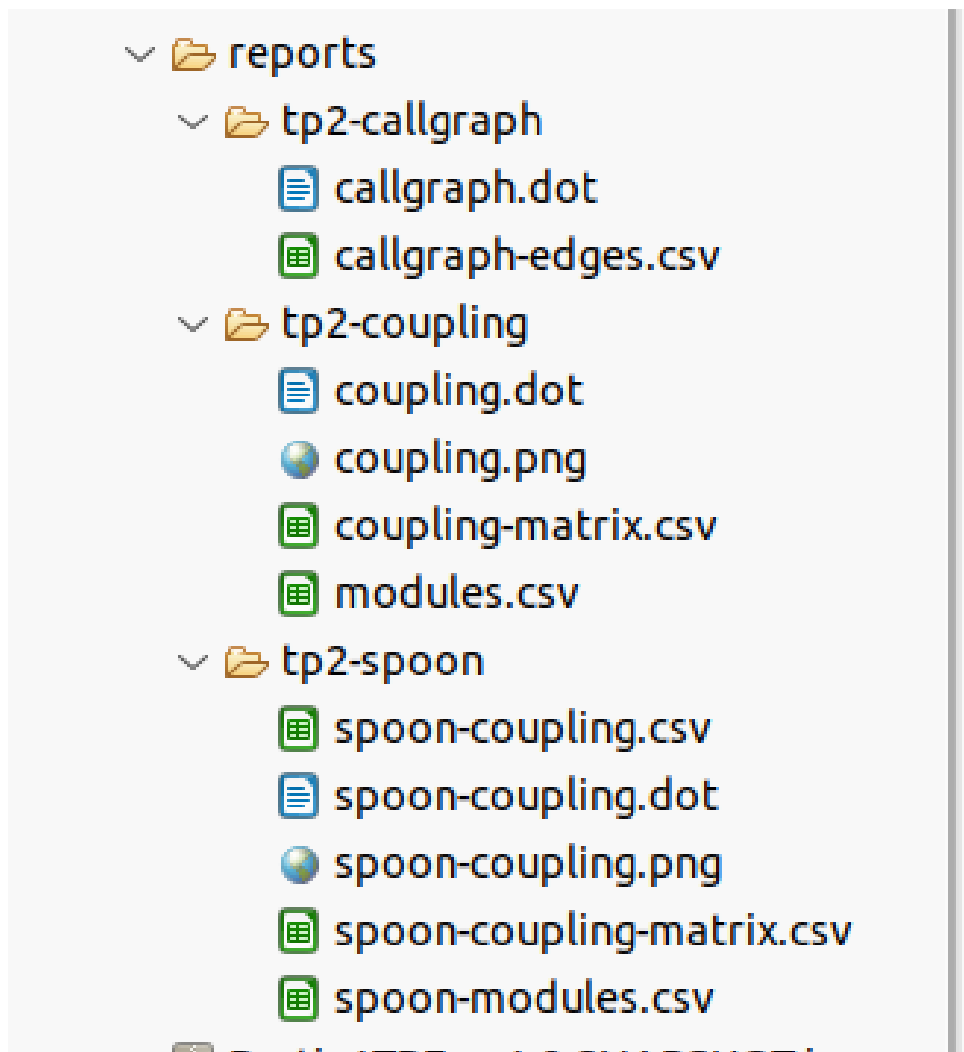
✓ Clustering terminé. Nombre final de modules : 4
Modules identifiés :
- tp2demo.Bike
- tp2demo.Car
- tp2demo.Engine
- tp2demo.Person

Vérification des moyennes intra-modules :
Module tp2demo.Bike Moyenne intra = 0,0000
Module tp2demo.Car Moyenne intra = 0,0000
Module tp2demo.Engine Moyenne intra = 0,0000
Module tp2demo.Person Moyenne intra = 0,0000
M=4, CP=0,010, maxClusters=2
• Meilleur couplage trouvé : tp2demo.Car ↔ tp2demo.Engine = 1,0000
• Fusion : tp2demo.Car + tp2demo.Engine → tp2demo.Car+tp2demo.Engine
• Meilleur couplage trouvé : tp2demo.Person ↔ tp2demo.Bike = 1,0000
• Fusion : tp2demo.Person + tp2demo.Bike → tp2demo.Person+tp2demo.Bike

✓ Clustering terminé. Nombre final de modules : 2
Modules identifiés :
- tp2demo.Car+tp2demo.Engine
- tp2demo.Person+tp2demo.Bike

Vérification des moyennes intra-modules :
Module tp2demo.Car+tp2demo.Engine Moyenne intra = 0,5000
Module tp2demo.Person+tp2demo.Bike Moyenne intra = 0,5000
```

## A.2 – Structure finale des rapports



## A.3 – Vérification du filtrage Spoon

```
=== Graphe de couplage via Spoon (filtré) ===  
tp2demo.Person → [tp2demo.Car, tp2demo.Bike]  
tp2demo.Car → [tp2demo.Engine]  
Matrice exportée : spoon-coupling-matrix.csv  
Modules exportés : spoon-modules.csv
```

# Liens du projet 'GitHub et Google drive'

## Lien du dépôt GitHub

Le code source complet du projet, incluant toutes les classes Java, les scripts d'analyse, ainsi que le rapport LaTeX, est disponible sur le dépôt GitHub suivant :

[Accéder au dépôt GitHub du projet TP2 – Évolution et restructuration du logiciel](#)

## Lien de la vidéo de démonstration

Une vidéo de démonstration détaillant les étapes d'exécution du projet (chargement du graphe, génération des matrices, clustering et résultats Spoon) est disponible à l'adresse suivante :

[Voir la vidéo de démonstration sur Google Drive](#)

Ces ressources complètent ce rapport et permettent de vérifier la bonne exécution du code, la structure du projet, et la conformité des résultats obtenus avec les objectifs du TP.