Université De Montpellier
Faculté Des Sciences

Niveau : Master 2

Module : Évolution et restructuration des logiciels

HAI913I

Rapport de TP N°3 : Compréhension des programmes

Supervisé par :
M. Abdelhak-Djamel Seriai
M. Marianne Huchard

Réalisé par :
Ahmed Kaci
Yanis Allouch

2021/2022

Table des matières

1 Introduction

Dans ce TP, ayant pour thème la compréhension des logiciels, nous allons réaliser une application permettant d'analyser le code source d'un programme java en se basant sur son arbre de syntaxe abstraite (AST). D'une part, nous manipulerons cette AST en utilisant Eclipse JDT ASTParser dont le fonctionnement se base sur le patron de conception Visiteur. D'autre part, nous utiliserons la librairie Spoon [1] développée par l'INRIA.

Tout d'abord, nous présenterons le travail effectué sur le graphe de couplage entre classes.
Puis, nous expliciterons la réalisation de l'identification de modules.
Par la suite, nous détaillerons sommairement le travail réimplémenté en Spoon.
Enfin, nous terminerons avec une partie d'analyse des temps d'exécution.

> Lien du dépôt GitLab du projet est le suivant :
> `https://gitlab.com/kaciahmed3/tp_comprension_logiciel_hai913i`. Un accès Reporter au compte GitLab @rechercheseriai a aussi été fourni.

Par ailleurs, les applications de tests et leurs résultats, utilisés dans ce rapport, sont disponibles sur ce dépôt GitLab public et ce tableur Google Sheets public fait par nos soins permettant une agrégation des résultats.

L'application réalisée prend en argument le chemin vers le dossier src contenant le programme java à analyser. Et cela s'applique dans tous les cas abordés dans ce rapport.

2 Graphe de couplage entre classes

Dans cet exercice, nous avons implémenté une métrique permettant de mesurer le couplage entre deux classes d'un programme. Puis, nous avons généré un graphe de couplage en appliquant cette métrique sur l'ensemble des classes de l'application analysé. La définition de cette métrique indiquée dans le sujet du TP, est la suivante :

— Nombre de relations (relation = appel) entre les couples de méthodes appartenant respectivement aux deux classes en question (A.mi et B.mj) / nombre de toutes les relations (binaire) entre les couples de méthodes appartenant respectivement à n'importe quelles deux classes de l'application analysée.

2.1 Calcul de la métrique de couplage

Nous avons choisi de l'implémenter de la façon suivante :

1. Nous récupérons l'ensemble des classes obtenues en utilisant le patron de conception visiteur,
2. Pour chaque classe, nous visitons chaque méthode déclarée,
3. Pour toutes méthodes déclarées, nous récupérons les invocations qu'elle effectue vers d'autres méthodes,
4. Pour chaque méthode invoquée, nous utilisons une résolution de type pour obtenir le nom de sa classe de déclaration,
5. Puis nous calculons le nombre total de relations entre toutes les classes ainsi que le nombre de relations entre chaque deux classes de l'application,
6. Enfin, nous calculons la métrique de couplage selon la définition rappelée précédemment.

2.2 Génération du graphe de couplage

Nous avons réalisé deux types de graphe de couplage pondéré ; un graphe ayant des relations orientées fournissant des informations détaillées sur le couplage entre les classes et un graphe non orienté qui sera utilisé pour l'identification de modules. Les graphes de couplages sont généré sous forme d'image PNG en utilisant du code DOT généré automatiquement par l'application après avoir effectué le parsing et le calcule du couplage entre les classes de l'application.
La figure 1 et figure 2 sont des échantillons du résultat fourni par le parsing de l'application disposé sur ce lien GitLab public.

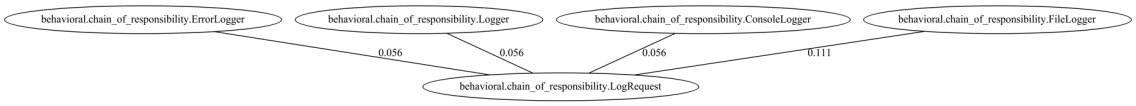

FIGURE 1 – Graphe de couplage bidirectionnel pondéré de l'application

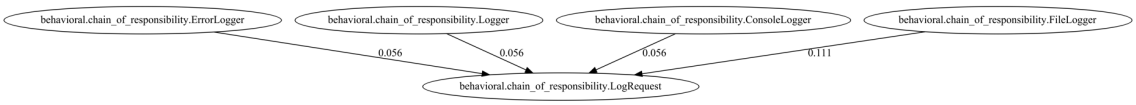

FIGURE 2 – Graphe de couplage unidirectionnel pondéré de l'application

3 Identification de Modules

Dans cet exercice, nous avons implémenté une identification de modules via la métrique basée sur le couplage entre deux classes précédent. La définition des algorithmes indiquée dans le sujet, sont les suivantes :

```
Clustering_hierarchique (fichiers code) : Arbre dendro;

classes := extraction_information(code);
clusters = classes;
tant que |clusters| > 1:
    (c1, c2) := clusterProche(clusters);
    c3 := Cluster(c1, c2);
    enleve(c1, clusters);
    enleve(c2, clusters);
    ajoute(c3, clusters);
fin tant que;
dendro := get(0, clusters);
retour dendro;
```

Listing 1 – Algorithme n°1 du clustering hiérarchique

```
Selection_clusters (arbre dendro) : Partition R;

Pile parcoursClusters;
empile(racine(dendro), parcoursClusters);
tant que !vide(parcoursClusters):
    Cluster pere = depile(parcoursClusters);
    Cluster f1 = fils1(pere, dendro);
    Cluster f2 = fils2(pere, dendro);
    si S(pere) > moyenne(S(f1, f2)):
        ajouter(pere, R);
    sinon:
        empile(f1, parcoursClusters);
        empile(f2, parcoursClusters);
    fin si;
fin tant que;
retour R;
```

Listing 2 – Algorithme n°2 du partionnement

Nous avons joint en annexe de ce TP la trace d'exécution permettant d'afficher les résultats obtenu à partir de l'application de l'algorithme de regroupement (clustering) hiérarchique sur le même projet cité précédemment.
Nous avons aussi joint en annexe la trace d'exécution permettant d'afficher les résultats obtenu à partir de l'application de l'algorithme d'identification des groupes de classes couplées.

4 Spoon pour l'identification de modules

Nous avons réimplémenté l'exercice 1 et 2 en utilisant cette fois-ci la bibliothèque Spoon. Ce dernier permet d'analyser et de transformer un code source Java.

L'implémentation en Spoon est similaire que l'implémentation réalisée pour l'exercice 1 et 2 avec l'ASTParser. Cependant, les résultats obtenus par l'analyse de la même application via Spoon donnent des résultats légèrement différents. Par exemple, la figure 1 et la figure 3 présente des résultats qui ne sont pas totalement identique alors qu'ils résultent d'une même application analysée (design pattern behavioral light).

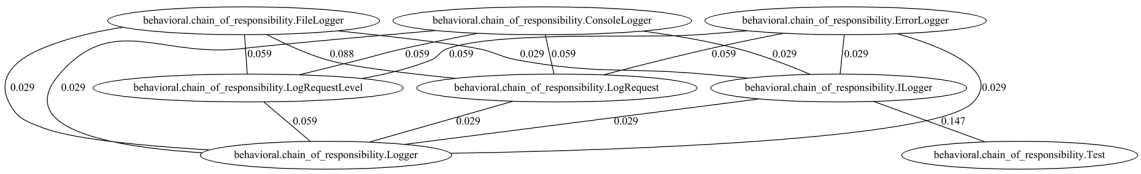

FIGURE 3 – Graphe de couplage bidirectionnel pondéré de l'application par Spoon

Les différences sont les suivantes :

1. Plus de type reconnu,
2. Des valeurs de couplages différentes.

La première différence, provient du fait que Spoon dispose d'un métamodèle plus complet et plus détaillé de Java. Ainsi, quand nous récupérons les types déclaratifs (statiquement) d'une application, nous récupérons aussi les interfaces, énumérations, etc. définies dans l'application analysée.
La différence entre les valeurs de couplages, est une conséquence du fait de la reconnaissance d'un plus grand nombre de types. Cependant, il est aussi possible que la résolution de type par ASTParser ne soit pas effective à 100%.

5 Analyse des temps d'éxécution

Nous remarquons une différence non triviale du temps d'exécution via l'ASTParser et Spoon. Dans cette section, nous allons donc présenter les résultats des temps d'exécutions obtenus à partir des traces d'exécutions sur trois versions d'un même programme disposé sur ce dépôt GitLab. Les résultats présentés ci-après, sont aussi disponibles dessus et sur cette feuille de calcul.

Nous avons posé manuellement des sondes dans notre code permettant de capturer les temps d'exécutions des implémentations des exercices présentés dans ce rapport. Pour un total de trente résultats agrégés dans la feuille de calcul précédente.
Les résultats ont été obtenus sur un processeur **AMD Ryzen 5 2600X (2018) 6 Cœurs OC à 4 GHz** pendant les tests. Plusieurs exécutions ont été faites, les ordres de grandeurs sont similaires.

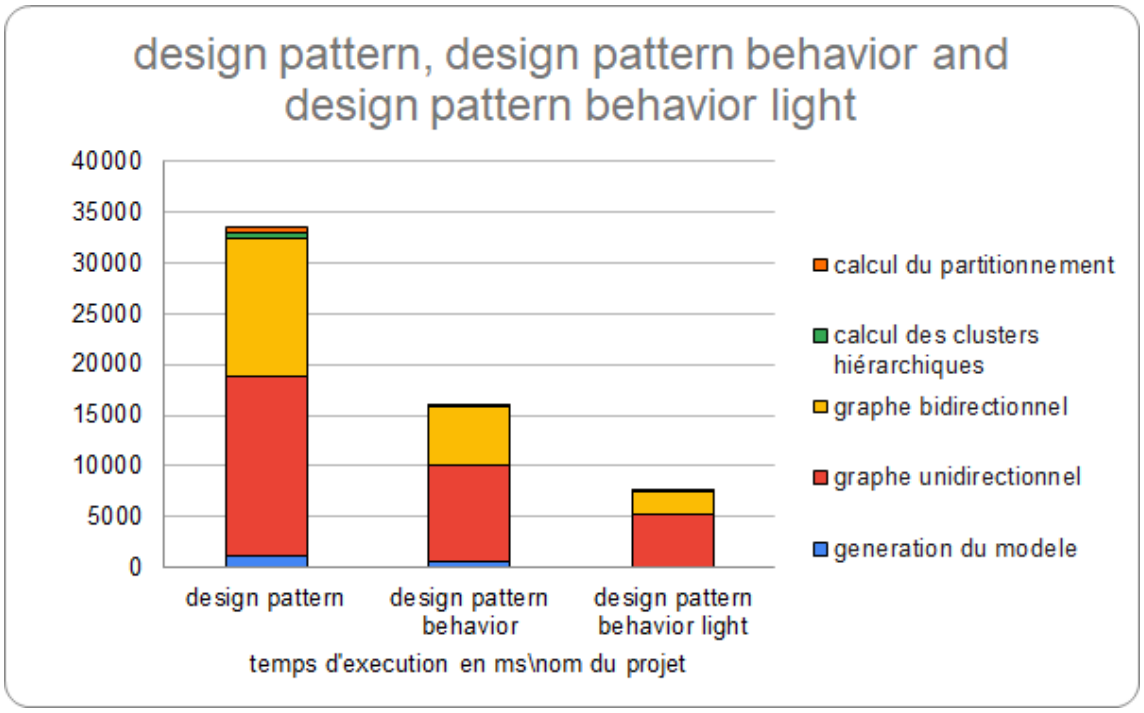

FIGURE 4 – Histogramme des temps d'exécutions cumulés par projet via ASTParser

La première observation des figure 4 et figure 5, nous permet de dire que le temps d'exécution croit a priori de manière exponentielle. Cette observation, nous permet de passer à la suivante, en effet, on observe que notre implémentation prend beaucoup de temps de calcul (significativement des autres métriques) sur le graphe de couplage (inhérent à la complexité algorithmique en $O(n^4)$).
De plus, une observation attentive de la figure 5 permet de se rendre compte de la différence des résultats abordés dans la section précédente. C'est-à-dire que Spoon, grâce a un métamodèle plus complet, nous fournit davantage d'informations à traiter.

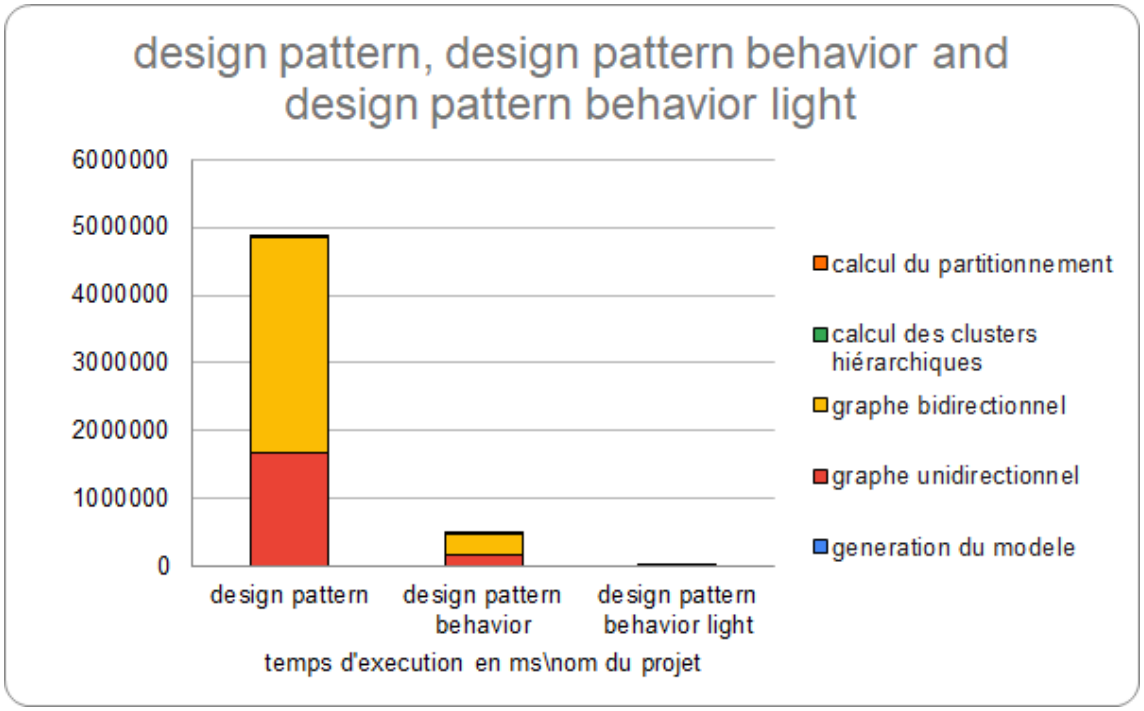

FIGURE 5 – Histogramme des temps d'exécutions cumulés par projet avec Spoon

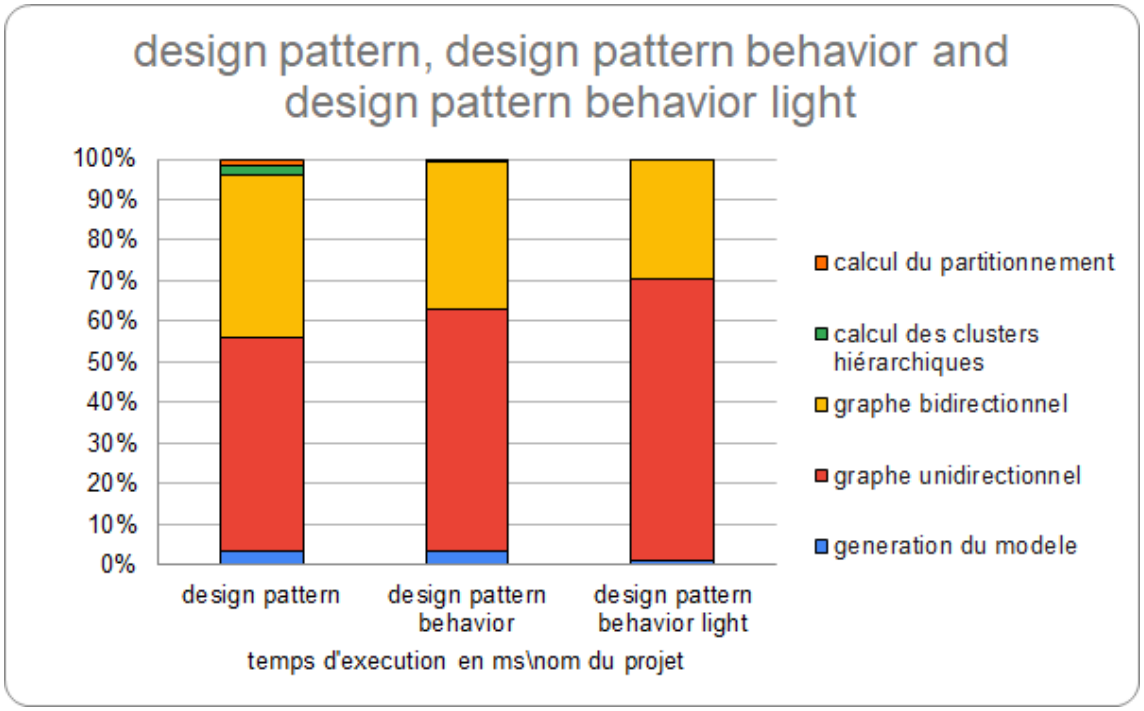

FIGURE 6 – Histogramme des proportions des temps d'exécutions cumulés par projet via ASTParser

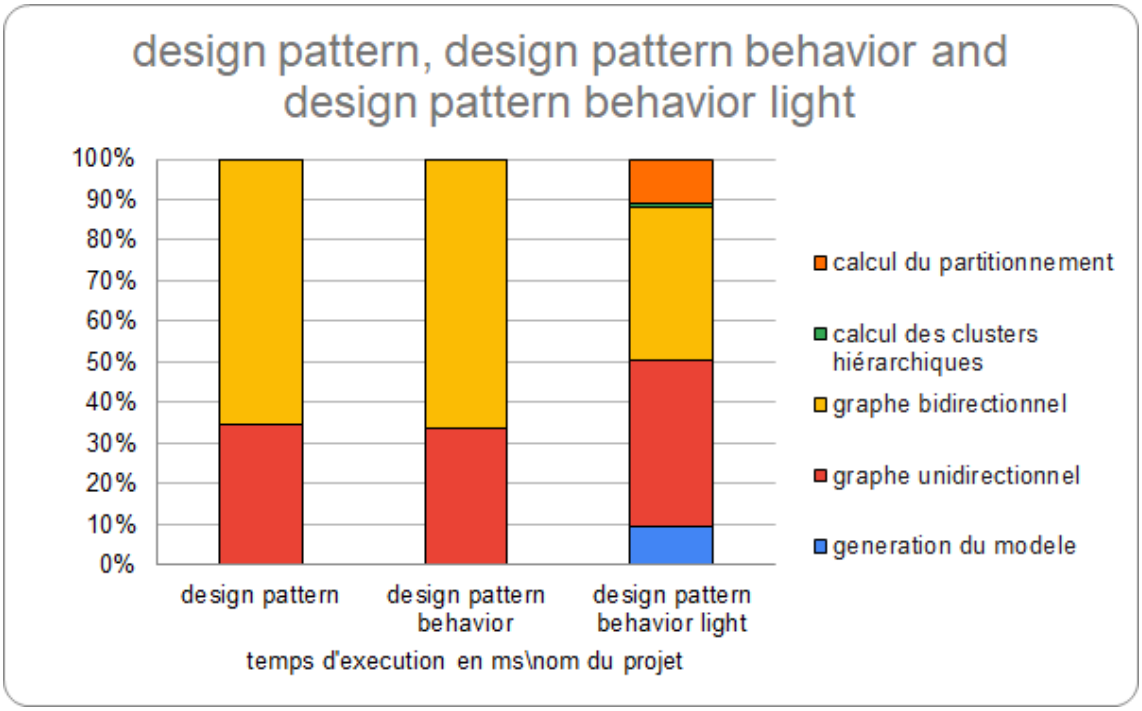

FIGURE 7 – Histogramme des proportions des temps d'exécutions cumulés par projet avec Spoon

Enfin, on observe d'une part sur la figure 7 que l'implémentation en Spoon passe significativement plus de temps de calcul sur le graphe bidirectionnel, et que sur la figure 6, l'implémentation ASTParser passe significativement plus de temps de calcul sur le graphe unidirectionnel. D'une autre part, on observe dans tous les cas que le temps de parsing par les librairies et le temps d'exécution de l'identification des modules est négligeable.

6 Conclusion

Dans ce TP, nous avons utilisé Eclipse JDT ASTParser et Spoon pour réaliser une application permettant l'analyse statique d'un programme. En se basant sur les résultat de cette dernière, nous avons calculé un ensemble d'informations statistiques qui ont permis par la suite d'être utilisé pour l'implémentation de la métrique de couplage entre classes. En outre, nous avons construit le graphe de couplage pondéré de l'application analysée. A partir des informations de ce graphe de couplage pondéré, nous avons implémenté un algorithme de clustering hierarchique. Puis implémenté un autre algorithme permettant d'identifier des groupes de classes couplées au sein de l'application analysée. Enfin, nous avons procédé à des relevés de temps d'exécution de notre application. Et nous avons constater, le caractère temporel allongé d'une analyse statique.

7 Annexe

7.1 Ressources utilisées

1. Documentation de l'API de ASTParser :
   — https://help.eclipse.org/latest/topic/org.eclipse.jdt.doc.isv/reference/api/org/eclipse/jdt/core/dom/package-summary.html.
2. Syntaxe de visualisation de graphe :
   — https://graphviz.org/,
3. Visualisation de graphe en ligne :
   — https://dreampuf.github.io/GraphvizOnline/.
4. Transformation Dot au format PNG en Java :
   — https://github.com/nidi3/graphviz-java.
5. Documentation de l'API Spoon :
   — https://spoon.gforge.inria.fr/mvnsites/spoon-core/apidocs/.
6. Site officiel de Spoon :
   — https://spoon.gforge.inria.fr/.

7.2 Traces d'exécution

Clustering hiérarchique

Trace d'exécution du clustering hiérarchique sur l'application disposé sur ce lien GitLab public.

```
Valeur des cluster lors de l'initialisation :
Classe : behavioral.chain_of_responsibility.Logger
valeur de la metrique de couplage de ce cluster : 0.0

Classe : behavioral.chain_of_responsibility.FileLogger
valeur de la metrique de couplage de ce cluster : 0.0

Classe : behavioral.chain_of_responsibility.ErrorLogger
valeur de la metrique de couplage de ce cluster : 0.0

Classe : behavioral.chain_of_responsibility.LogRequest
valeur de la metrique de couplage de ce cluster : 0.0

Classe : behavioral.chain_of_responsibility.Test
valeur de la metrique de couplage de ce cluster : 0.0

Classe : behavioral.chain_of_responsibility.ConsoleLogger
valeur de la metrique de couplage de ce cluster : 0.0


```

```
Creation de la hierarchie des clusters :

Fusion entre clusters
 Partie A De la fussion :
Classe :  behavioral.chain_of_responsibility.FileLogger
Parie B de la fusion
Classe :  behavioral.chain_of_responsibility.LogRequest
valeur du couplage : 0.111



Liste des cluster de cette etape:

 Cluster :
Classe : behavioral.chain_of_responsibility.Logger
valeur de la metrique de couplage de ce cluster : 0.0


 Cluster :
Classe : behavioral.chain_of_responsibility.ErrorLogger
valeur de la metrique de couplage de ce cluster : 0.0


 Cluster :
Classe : behavioral.chain_of_responsibility.Test
valeur de la metrique de couplage de ce cluster : 0.0


 Cluster :
Classe : behavioral.chain_of_responsibility.ConsoleLogger
valeur de la metrique de couplage de ce cluster : 0.0


 Cluster :
Classe : behavioral.chain_of_responsibility.LogRequest
Classe : behavioral.chain_of_responsibility.FileLogger
valeur de la metrique de couplage de ce cluster : 0.111


Fusion entre clusters
 Partie A De la fussion :
Classe :  behavioral.chain_of_responsibility.Logger
Parie B de la fusion
Classe :  behavioral.chain_of_responsibility.LogRequest
Classe :  behavioral.chain_of_responsibility.FileLogger
valeur du couplage : 0.056



Liste des cluster de cette etape:

 Cluster :
Classe : behavioral.chain_of_responsibility.ErrorLogger
```

```
valeur de la metrique de couplage de ce cluster : 0.0


 Cluster :
Classe : behavioral.chain_of_responsibility.Test
valeur de la metrique de couplage de ce cluster : 0.0


 Cluster :
Classe : behavioral.chain_of_responsibility.ConsoleLogger
valeur de la metrique de couplage de ce cluster : 0.0


 Cluster :
Classe : behavioral.chain_of_responsibility.LogRequest
Classe : behavioral.chain_of_responsibility.Logger
Classe : behavioral.chain_of_responsibility.FileLogger
valeur de la metrique de couplage de ce cluster : 0.056


Fusion entre clusters
 Partie A De la fussion :
Classe :  behavioral.chain_of_responsibility.ErrorLogger
Parie B de la fusion
Classe :  behavioral.chain_of_responsibility.LogRequest
Classe :  behavioral.chain_of_responsibility.Logger
Classe :  behavioral.chain_of_responsibility.FileLogger
valeur du couplage : 0.056



Liste des cluster de cette etape:

 Cluster :
Classe : behavioral.chain_of_responsibility.Test
valeur de la metrique de couplage de ce cluster : 0.0


 Cluster :
Classe : behavioral.chain_of_responsibility.ConsoleLogger
valeur de la metrique de couplage de ce cluster : 0.0


 Cluster :
Classe : behavioral.chain_of_responsibility.ErrorLogger
Classe : behavioral.chain_of_responsibility.LogRequest
Classe : behavioral.chain_of_responsibility.Logger
Classe : behavioral.chain_of_responsibility.FileLogger
valeur de la metrique de couplage de ce cluster : 0.056


Fusion entre clusters
 Partie A De la fussion :
```

```
Classe :  behavioral.chain_of_responsibility.ConsoleLogger
Parie B de la fusion
Classe :  behavioral.chain_of_responsibility.ErrorLogger
Classe :  behavioral.chain_of_responsibility.LogRequest
Classe :  behavioral.chain_of_responsibility.Logger
Classe :  behavioral.chain_of_responsibility.FileLogger
valeur du couplage : 0.056



Liste des cluster de cette etape:

 Cluster :
Classe : behavioral.chain_of_responsibility.Test
valeur de la metrique de couplage de ce cluster : 0.0


 Cluster :
Classe : behavioral.chain_of_responsibility.ErrorLogger
Classe : behavioral.chain_of_responsibility.LogRequest
Classe : behavioral.chain_of_responsibility.Logger
Classe : behavioral.chain_of_responsibility.ConsoleLogger
Classe : behavioral.chain_of_responsibility.FileLogger
valeur de la metrique de couplage de ce cluster : 0.056



 ********** liste Final des clusters **********


Affichage de la hierarchie des clusters
nouveau Cluster
Classe : behavioral.chain_of_responsibility.Test
valeur de la metrique de couplage de ce cluster : 0.0



nouveau Cluster
Classe : behavioral.chain_of_responsibility.ErrorLogger
Classe : behavioral.chain_of_responsibility.LogRequest
Classe : behavioral.chain_of_responsibility.Logger
Classe : behavioral.chain_of_responsibility.ConsoleLogger
Classe : behavioral.chain_of_responsibility.FileLogger
valeur de la metrique de couplage de ce cluster : 0.056



Temps d'execution de calcul des clusters avec ASTParser : 4 Ms
```

Listing 3 – Trace de l'algorithme n°1 du clustering hiérarchique

Identification des modules

Trace d'exécution de l'identification des modules sur l'application disposé sur ce lien GitLab public.

```
Temps d'execution de la generation du model par ASTParser : 60 Ms
Temps d'execution de generation du graphe de couplage
    bidirectionnel avec ASTParser : 1 Ms
 Valeur des cluster lors de l'initialisation :
Classe : behavioral.chain_of_responsibility.ErrorLogger
valeur de la metrique de couplage de ce cluster : 0.0

Classe : behavioral.chain_of_responsibility.Test
valeur de la metrique de couplage de ce cluster : 0.0

Classe : behavioral.chain_of_responsibility.FileLogger
valeur de la metrique de couplage de ce cluster : 0.0

Classe : behavioral.chain_of_responsibility.ConsoleLogger
valeur de la metrique de couplage de ce cluster : 0.0

Classe : behavioral.chain_of_responsibility.Logger
valeur de la metrique de couplage de ce cluster : 0.0

Classe : behavioral.chain_of_responsibility.LogRequest
valeur de la metrique de couplage de ce cluster : 0.0


Creation de la hierarchie des clusters :

Construction des partitions

 Couplage du pere : 0.056

 moyenne des couplages des fils : 0.028

On ajoute le pere a la partition

 Partitions Construites a cette etape :
Classe : behavioral.chain_of_responsibility.ErrorLogger
Classe : behavioral.chain_of_responsibility.LogRequest
Classe : behavioral.chain_of_responsibility.Logger
Classe : behavioral.chain_of_responsibility.ConsoleLogger
Classe : behavioral.chain_of_responsibility.FileLogger
valeur de la metrique de couplage de ce cluster : 0.056


 Partitions Construites a cette etape :
Classe : behavioral.chain_of_responsibility.ErrorLogger
Classe : behavioral.chain_of_responsibility.LogRequest
Classe : behavioral.chain_of_responsibility.Logger
Classe : behavioral.chain_of_responsibility.ConsoleLogger
Classe : behavioral.chain_of_responsibility.FileLogger
```

```
valeur de la metrique de couplage de ce cluster : 0.056


 Partitions Construites a cette etape :
Classe : behavioral.chain_of_responsibility.ErrorLogger
Classe : behavioral.chain_of_responsibility.LogRequest
Classe : behavioral.chain_of_responsibility.Logger
Classe : behavioral.chain_of_responsibility.ConsoleLogger
Classe : behavioral.chain_of_responsibility.FileLogger
valeur de la metrique de couplage de ce cluster : 0.056


 Partitions Construites a cette etape :
Classe : behavioral.chain_of_responsibility.ErrorLogger
Classe : behavioral.chain_of_responsibility.LogRequest
Classe : behavioral.chain_of_responsibility.Logger
Classe : behavioral.chain_of_responsibility.ConsoleLogger
Classe : behavioral.chain_of_responsibility.FileLogger
valeur de la metrique de couplage de ce cluster : 0.056



 ************** liste Final des partitions ***************


 partition :
Classe : behavioral.chain_of_responsibility.ErrorLogger
Classe : behavioral.chain_of_responsibility.LogRequest
Classe : behavioral.chain_of_responsibility.Logger
Classe : behavioral.chain_of_responsibility.ConsoleLogger
Classe : behavioral.chain_of_responsibility.FileLogger
Nombre de partitions : 1
Temps d'execution de calcul des partitions avec ASTParser : 2 Ms
```

Listing 4 – Trace de l'lgorithme n°2 du partionnement

Références bibliographiques

[1] Renaud Pawlak et al. « Spoon : A Library for Implementing Analyses and Transformations of Java Source Code ». In : *Software : Practice and Experience* 46 (2015), p. 1155-1179. DOI : 10.1002/spe.2346. URL : https://hal.archives-ouvertes.fr/hal-01078532/document.

[2] Abdelhak-Djamel Seriai. *Évolution et maintenance des systèmes logiciels*. Paris : Hermès Science publications Lavoisier, 2014. ISBN : 9782746245549.