TP1 : Analyse statique
Évolution et restructuration des logiciels

Mohamad Satea Almallouhi - Tony Nguyen
M1 Génie Logiciel
Faculté des Sciences
Université de Montpellier.

6 octobre 2024

Résumé

Rapport d'exercice sur l'analyse statique

Table des matières

Introduction

Dans le cadre de l'Unité d'Enseignement Évolution et restructuration des logiciels, nous allons analysés un programme en observant son code source de manière statique. L'étape d'extraction des informations a été réalisé précédement. Nous nous trouvons à présent dans l'étape de traitement des propriétés dans le workflow. Nous allons étudier le concept de couplage des classes.

Tout d'abord, à partir du travail précédent, nous allons nous servir du graph d'appel des méthodes écrites dans les classes du projet analysé. Cela nous permettra de calculer le couplage entre les différentes classes.

Ainsi, grâce au graph de couplage, nous allons partitioné notre ensemble de classes en différent modules.

Démonstration vidéo

En ligne sur Youtube, à l'adresse URL une démonstration vidéo de notre travail.

Installation

Vous trouverez les instructions dans le README.md

To Do
— add code picture
— add resultat picture
— diagram class visiteur
— diagram class de l'app
— definition du couplage avec écritures math
— tous les algo en latex stylé
— →!!! → vidéo ←!!! ←
Faire une vidéo, le rapport avec des screenshot des résultats et du code et enfin un read.md(instruction). En plus, pour le bonus, faire une belle application, des tests unitaires, faire le rapport en Latex.

1 UML ! ! ! !

2 Graphes

Nous nous intéressons au couplage entre les classes de notre application. Il serait logique de réunir les classes fortement dépendantes les unes des autres. De la même façon, les classes qui n'ont aucun rapport entre elle, pour des raisons de clarté, peuvent être isolé.

2.1 Algorithmes de création du graphe d'appel

La construction de ce graph est la base de ce travail. Il nous permettra ensuite de calculer le couplage entre les classes ...

Les méthodes qui se surchage entre elles (les méthodes ayant le même nom dans une classe mais avec une signature différente) sont confondues.

2.2 Le couplage

Définition TODO INSERT EPIC MATH FORMULA HERE

2.3 Graphe de couplage inter-classes

UML TODO INSERT EPIC CLASS DIAGRAM HERE

2.4 Montrer les visiteur

? ? ?

3 Clusturing

Nous allons maintenant voir comment nous avons rassemblé les classes entre elles.

3.1 Algorithme de clustering

Dendrogramme

3.2 Identification des modules

Partitionnement des classes

4 Spoon

Algorithm 1: An algorithm to make call graph

```
Data: programme
Result: graphe d'appel
g ← new Graph();
for each class c do
    for each method m implemented in c do
        for each method invoqued i in m do
            if !isExternalMethod(i) then
                g.addSommet(m);
                g.addSommet(i) ;                    /* ajout des sommets m et i */
                g.addVertex(m, i) ;                 /* ajoute une arete de m vers i */
            end
        end
    end
end
return g;
```

Algorithm 2: An algorithm to make a weighted graph corresponding to the coupling between class

```
Data: le graphe d'appel
Result: graphe pondere
graphPondere ← new GraphPondere();
/* Produit cartésion entre toutes les classes de l'application en retirant les
   couples identiques (x,x)                                                    */
for each String aClassName1 in classNameSet do
    for each String aClassName2 in classNameSet do
        if aClassName1 != aClassName2 then
            cpValue ← calculCouplageValueEntre(aClassName1, aClassName2);
            if cpValue > 0 then
                graphPondere.addArete(aClassName1, aClassName2, cpValue);
            end
        end
    end
end
return graphPondere;
```

Algorithm 3: Clustering algorithm (Creating the Dendrogramme)

UML INSERT UML CLASS DIAGRAM OF Cluster composite pattern
Data: *graph d'appel*
Result: *the dendrogramme*
`/* Stratégie: toutes les class sont leurs propres cluster. On va essayer de fusioner les clusters entre eux en fonction du couplage */`
clusters ← *Cluster*[];
for *each String aClassName in classNameSet* **do**
  clusters.add(*new Leaf*(*aClassName*));
end
i ← 0;
Noderoot ← *null*;
while *i* < *size(clusters)* **do**
  n ← *laFusionEntreLes*2*CLustersLesPlusProche*();
  on retire les 2 *enfants du noeud n*;
  clusters.add(*n*);
  i + +;
end
return root;

Algorithm 4: Identification des modules

Data: *dendrogramme root*
Result: *Partitionnement en modules*
result`/* une liste d'ensemble de String */`
auxGetModule(*root*, *result*);
returnresult;

Algorithm 5: auxGetModule

Data: *dendrogramme root*
Result: *Partitionnement en modules*
if *root possède une valeur de couplage suffisante ou si c'est une feuille* **then**
  on a trouvé un module
else
  on fait la même chose au enfants de root de façon récursive
end