HAI913I
Evolution et restructuration des logiciels

TP1 - Rendu d'exercice

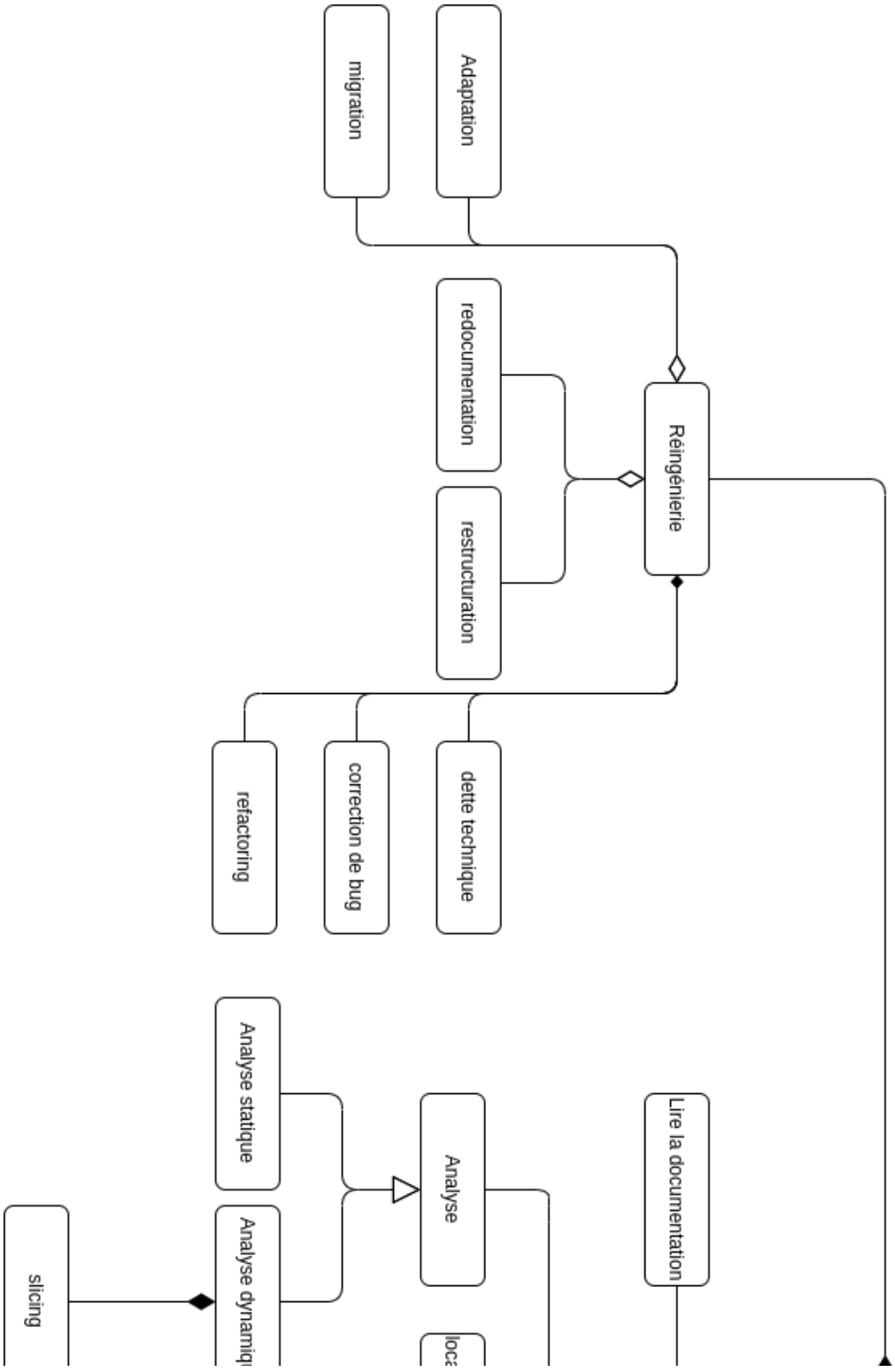
Auteur :

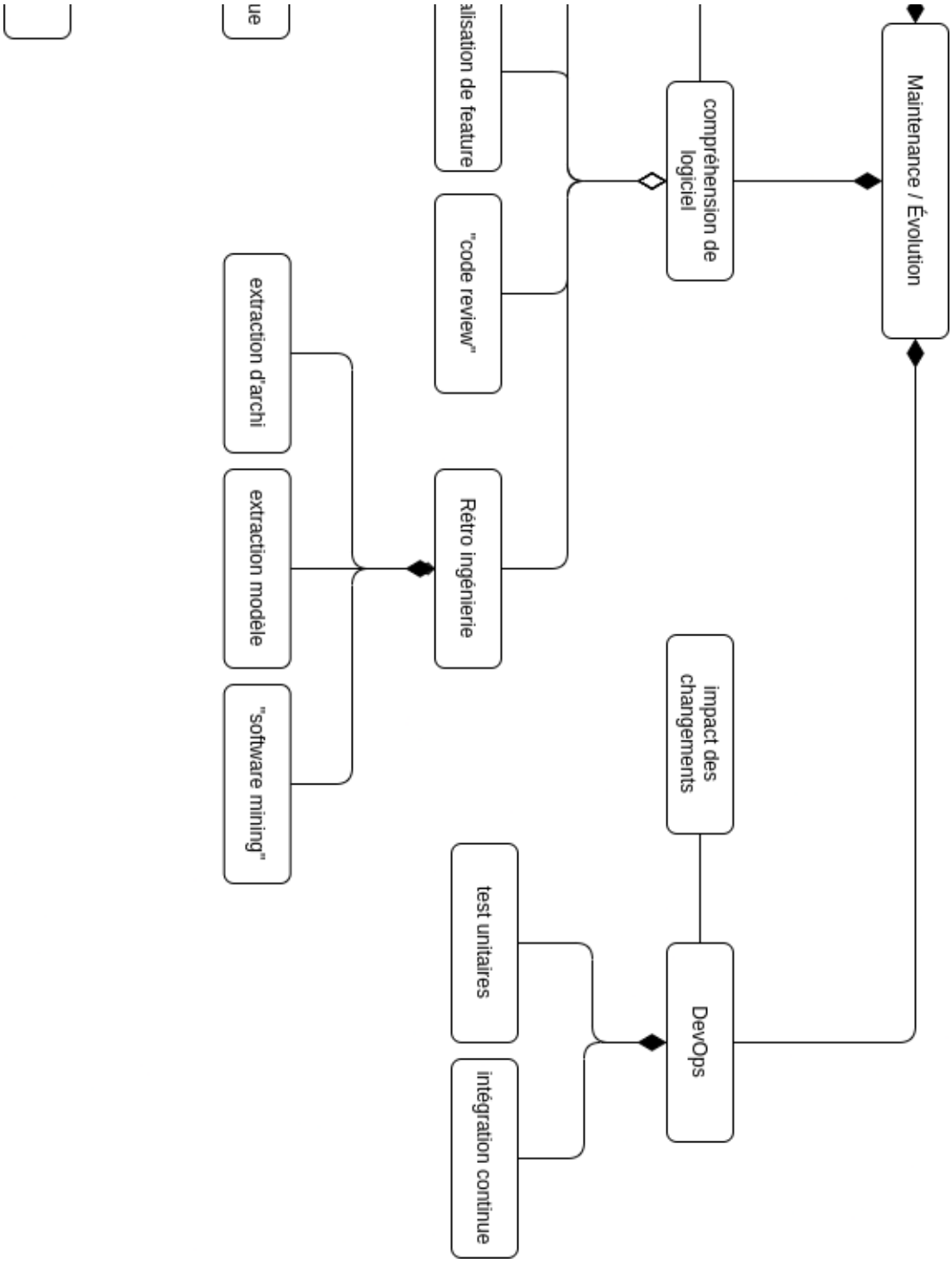
Canta Thomas (21607288)
Reiter Maxime (21604458)

Master 2 - Génie Logiciel
Faculté des sciences de Montpellier
Année universitaire 2021/2022

Exercice 1 - Compréhension des concepts liés à l'évolution/maintenance des logiciels

Vous trouverais l'UML ci-dessous au format image avec ce rapport ou alors vous pouvez directement y accéder en cliquant ici.





Exercice 3 - Outils de maintenance/évolution

✎ SonarQube

SonarQube est un logiciel open source permettant de nettoyer, optimiser et réparer un code afin de le rendre le plus propre possible en alertant aussitôt les développeurs pour prendre les mesures nécessaires. Très utile pour obtenir une application finale stable et faciliter le travail, notamment

pour la maintenabilité du logiciel. Il permet aux équipes de développement de perdre moins de temps à chercher et corriger les erreurs, rendant le processus plus efficace donc moins coûteux notamment pour ceux adoptant une approche Agile et DevOps

✎ CheckStyle

CheckStyle est un outil de développement pour faciliter les programmeurs à écrire du code Java qui respecte une norme de codage en automatisant le processus de vérification du code. Il est donc idéal pour les projets qui veulent faire respecter une norme, simple et rapide d'utilisation il s'installe comme un plugin sur son éditeur de texte. Il analyse en quelques secondes seulement notre projet et rapporte une expertise complète par le biais d'avertissements ou d'erreurs que même les meilleurs éditeur, comme IntelliJ [1], ne font pas transparaître. Conserver une norme tout au long d'un projet permet de maintenir efficacement et sur le long terme un logiciel.

Exercice 4

✎ Refactoring Asynchrony in JavaScript (voir pdf)

Ce document nous introduit le problème des fonctions de *callback* imbriqué en Javascript, aussi nommée "callback hell". Ce problème engendre une offuscation du code et une difficile gestion d'erreur. Pour pallier à ceci, un design pattern a été introduit, permettant de gérer les évènements asynchrones tout en conservant la norme "error-first" du Javascript. Ce design pattern, aujourd'hui très connus, a pour nom *Promise*.

La refactorisation de callback imbriqués en Promise est souvent réalisé de façon manuelle et prône donc à l'erreur, surtout lorsque le code est difficile à lire. C'est pourquoi l'étude, menée à travers ce document, cherche à résoudre ce procédé en automatisant ce processus.

Voici en nos mots comment fonctionne PromiseLand. Dans un premier temps PromiseLand détecte les fonctions de callback puis l'identifie. Dans un second temps vient la conversion en Promise, pour ce faire ils utilisent deux méthodes différentes, soit en modifiant la fonction elle-même où alors en englobant la fonction par une autre fonction qui retourne une Promise. Ces deux moyens de conversion sont choisis selon certains critères. Dans un troisième et dernier temps, une étape d'optimisation est opérer, si possible, en "aplatissant" les multiples fonctions de callback pour obtenir une fonction plus lisible.

L'évaluation de leur programme, PromiseLand, démontre que même sur plusieurs configurations ils trouvent 83% des fonctions de callback. Comparés à un programme concurrent comme *Dues*, ils ont su être plus efficace en dénichant 3 fois plus de fonction de callback à optimiser. De plus, leurs tests de performance montrent qu'en approximativement 3 secondes leur programme est capable de transformer un code source, ce qui est très peu sachant qu'il suffit de lancer PromiseLand une seule fois par code source.

1. Plus d'information sur IntelliJ : jetbrains.com/fr-fr/idea/