

Rapport de laboratoire

|  |  |
| --- | --- |
| **No de laboratoire** | 1 |
| **Étudiant(s)** | Sébastien Lago  Marc-André Allard  Israël Hallé |
| **Code(s) permanent(s)** | LAGS04128102  HALI17049101  ALLM09029106 |
| **Cours** | LOG430 |
| **Session** | Hiver 2014 |
| **Groupe** | 02 |
| **Professeur** | B. Galarneau |
| **Chargés de laboratoire** |  |
| **Date de remise** | 5 février 2014 |

Contents

[Introduction 3](#_Toc379327485)

[Implémentation 3](#_Toc379327486)

[Analyse architecturale 4](#_Toc379327487)

[Vue architecturale 4](#_Toc379327488)

[Système original 4](#_Toc379327489)

[Système final 7](#_Toc379327490)

[Critères adoptés pour le mappage des classes 9](#_Toc379327491)

[Interprétation de la matrice de dépendance 9](#_Toc379327492)

[Modification du nouveau système si elle accédait à une base de données 11](#_Toc379327493)

[Discussion 12](#_Toc379327494)

[Conclusion 13](#_Toc379327495)

[Annexe A 14](#_Toc379327496)

[Plan de test 14](#_Toc379327497)

# Introduction

Être capable de mapper l’implémentation orientée objet à une architecture précise est important pour pouvoir créer une architecture logicielle cohérente. Plusieurs outils facilitent la transition entre le code et une architecture, par exemple les diagrammes de classes et les matrices de dépendances. Dans la première partie de ce laboratoire, nous appliquerons des modifications au code. Ensuite, nous utiliserons les outils mentionnés précédemment pour analyser nos décisions de conceptions et créer une architecture en couche.

# Implémentation

Une fois l’architecture du système existant établie, nous avons ajouté les nouvelles fonctionnalités au code. Nous n’avons pratiquement pas eu à réécrire le code existant. Nous avons surtout rajouté des méthodes et des lignes de codes là où nécessaire. Une exception a été la création initiale des ressources. Dans le programme initial, chaque ressource a une liste de projets déjà assignée. Cependant, ces projets sont vides, mis à part leur titre. C’est problématique si l’on veut connaitre plus en détail les projets assignés à une ressource. Après modifications, la liste de projets déjà assignés dans chaque ressource est maintenant complète. Si cette omission n’était pas intentionnelle, elle serait à corriger pour l’itération future du cours. Une fois les trois modifications apportées, nous avons décidé de rajouter une classe de test (TestModifications), qui permet de valider les fonctionnalités ajoutées.

# Analyse architecturale

## Vue architecturale

### Système original

#### Matrice de dépendance

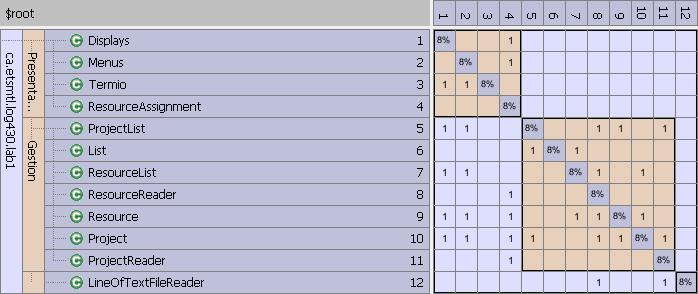


Figure 1

#### Diagramme de classes

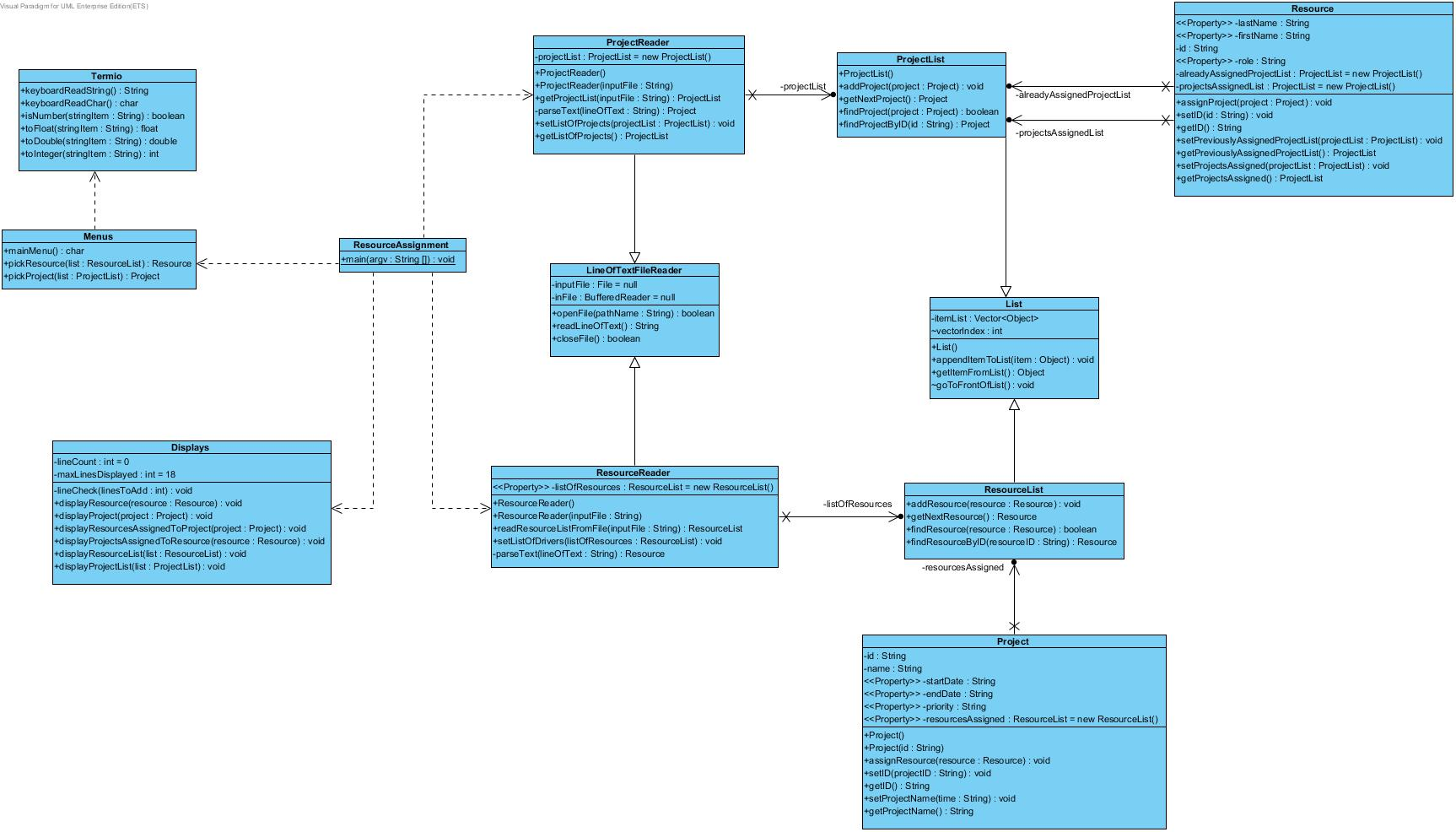


Figure 2

#### Mappage des classes aux couches de l’architecture



Figure 3

### Système final

#### Matrice de dépendance

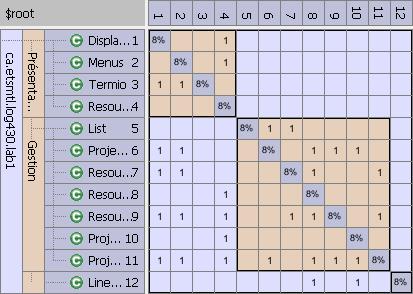


Figure 4

#### Vue architecturale du nouveau système

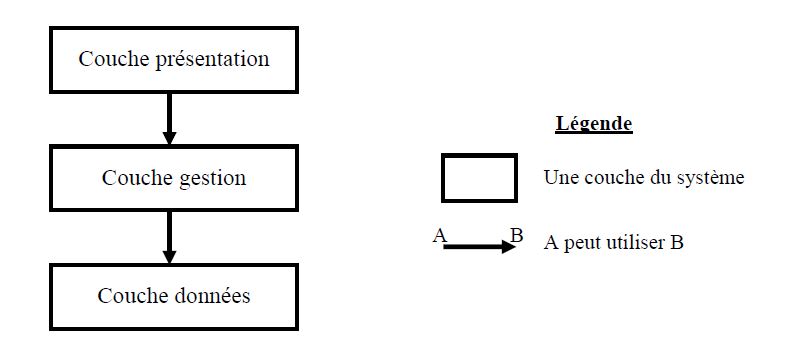


Figure 5

#### Diagramme de classes

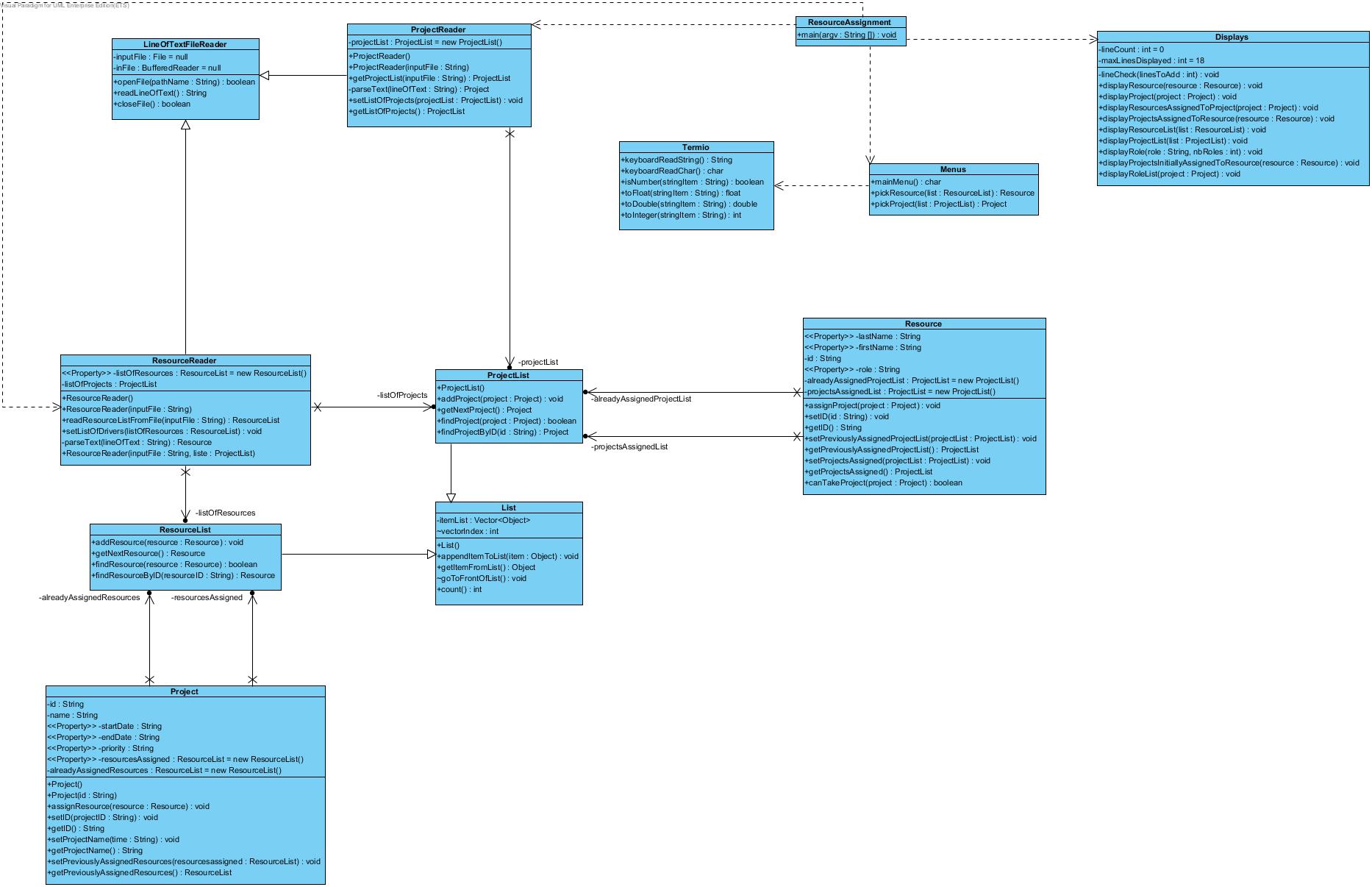


Figure 6

#### Mappage des classes aux couches de l’architecture

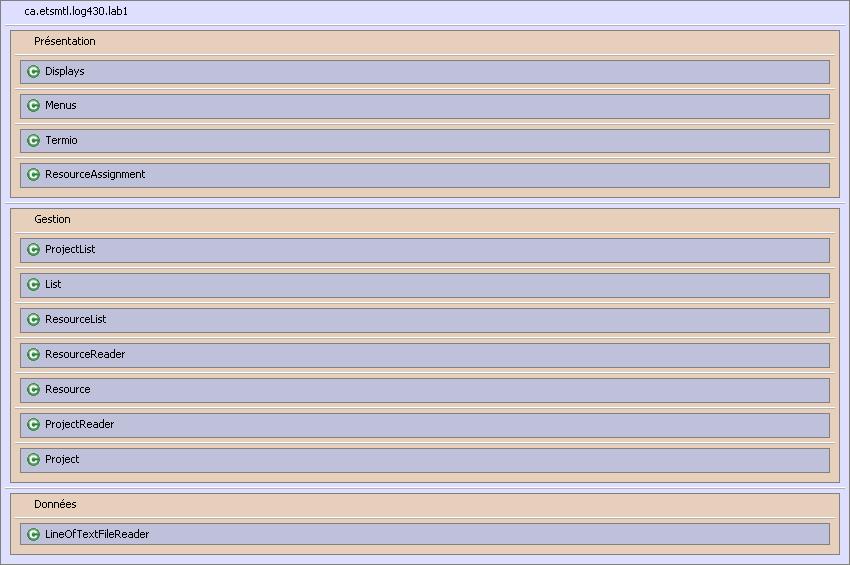


Figure 7

## Critères adoptés pour le mappage des classes

Nous avons choisi la classe « LineOfTextFileReader » comme la seule classe faisant partie de la couche « données », car c’est la classe qui accède aux données sur le disque tout en restant indépendante des classes de la couche Gestion. Elle lit les lignes dans un fichier texte et les retourne vers les classes de la couche supérieure (ProjectReader et ResourceReader) qui se chargeront d’interpréter chacune des lignes de texte et, par la suite, créer des listes de projets et de ressources. Les classes Project, ProjectList, Resource, ResourceList et List font partie de la même couche que ProjectReader et ResourceReader qui est la couche « Gestion ». Nous avons choisi de placer ces classes dans cette couche, car elles n’extraient pas directement les données de la source (des fichiers texte), mais elles contiennent les données provenant de celle-ci. Ces classes représentent la logique du domaine de l’application. Pour terminer, nous avons placé les autres classes dans la couche de présentation, car ce sont les classes qui interagissent directement avec l’utilisateur à partir d’entrées/sorties en console. La classe « Termio » lit ce que l’utilisateur entre dans l’invite de commande, la classe « Menus » présente des menus à l’utilisateur, la classe « Displays » affiche les résultats à l’utilisateur et la classe « ResourceAssignment » parce qu’elle utilise les classes de la couche présentation et s’occupe d’initialiser et charger les objets du domaine.

## Interprétation de la matrice de dépendance

La matrice s’avère fort utile pour valider la conformité d’un projet à son architecture. On peut facilement observer les dépendances de chacune des classes dans une grille relativement facile à lire. Dans le contexte d’une architecture par couches, on effectue d’abord le mappage des classes avec leur couche. Ensuite, dans la grille, on dispose les classes de la couche supérieure à la couche inférieure et l’on n’a qu’à s’assurer que tous les « un » dans la grille se situent sous la diagonale. Si c’est le cas, cela signifie qu’il n’y a aucune classe qui utilise une autre située dans une couche supérieure.

Pour notre système, la matrice de dépendance montre que chaque couche n’utilise que les classes de la couche directement en dessous. Il n’y a donc pas d’amélioration à apporter à notre système de ce côté.

## Modification du nouveau système si elle accédait à une base de données

Si notre nouveau système devait accéder à une base de données au lieu de fichiers texte, la classe à modifier serait LineOfTextFileReader, car le changement concerne la couche des données et c’est la seule à faire partie de cette couche. Nous appellerions cette classe DataReader. Elle posséderait des méthodes analogues à celles de LineOfTextFileReader (open au lieu d’openFile, readRow au lieu de readLineOfText et close au lieu de closeFile). Les classes « ResourceReader » et « ProjectReader » hériteraient de cette nouvelle classe au lieu de « LineOfTextFileReader » et les appels aux méthodes de l’ancienne classe seraient remplacés par ceux de la nouvelle classe. Toutes les autres classes, qui se retrouvent sur des couches supérieures, ne subiront aucun changement, car c’est le but du modèle par couches (apporter un changement une seule couche à la fois). Il est à noter qu’il serait préférable d’avoir nommé les méthodes de la classe LineOfTextFileReader de manière plus générique (Open à la place de OpenFile, Read, à la place de ReadFile). Les méthodes utilisées par des couches supérieures devraient être nommées de cette façon pour faciliter le remplacement d’une couche par une autre.

Voici le mappage des classes du nouveau système. Tout est pareil sauf le nom de la classe LineOfTextFileReader qui devient DataReader. La vue architecturale serait la même que l’ancienne, constituée d’une couche « Présentation », « Gestion » et « Données ».



Figure 8

## Discussion

Nous avons décidé de commencer le laboratoire en faisant l’analyse architecturale du système existant. Nous avons utilisé Visual Paradigm pour générer un diagramme de classe à partir du code. Ceci nous a permis de bien voir les relations entre les classes et les dépendances. Ensuite, nous avons utilisé l’outil Lattix pour créer la matrice de dépendance du système. Après analyse, nous avons conclu que les classes étaient séparées en 3 couches distinctes : Présentation, Gestion et Données. Nous n’étions pas sûres si la classe RessourceAssigment devait être dans la couche Présentation ou la couche Gestion. Finalement, nous l’avons mise dans la couche Présentation, étant donné qu’elle « utilise » les classes de présentation et les gère. La mettre dans Gestion créerait des dépendances d’une couche inférieure à une couche supérieure, ce qui est à proscrire dans ce modèle d’architecture. De plus, nous n’étions pas sûres si les classes ProjectReader et ResourceReader devaient être dans Gestion ou dans Données. Nous avons conclu qu’étant donné que ces deux classes utilisaient quelques classes de la couche Gestion et que la classe LineOfTextReader était la seule à interagir directement avec les fichiers, elle serait la seule à être dans la couche Donnée et les classes ProjectReader et ResourceReader seraient dans la couche de Gestion.

Finalement, nous avons comparé l’architecture du système initiale à celle du système final. Après avoir analysé la nouvelle matrice de dépendances, nous avons conclu qu’il n’y avait pas de modifications à apporter à l’architecture existante. La raison est que chaque couche n’utilise que la couche directement en dessous. Le modèle en couche est donc tout à fait respecté.

## Conclusion

Dans ce laboratoire, nous avons mappé un programme, écrit en java, à une architecture en couche. Nous avons ensuite modifié le code pour lui ajouter des fonctionnalités. Finalement, nous avons vérifié que l’architecture était toujours valide, une fois le code modifié.

# Annexe A

# Plan de test

Nous avons créé une classe de test unitaire qui permet de tester l’application. Pour ce faire, le dossier « test » fourni avec l’application est requis. Ce dossier contient les fichiers projects.txt et resouces.txt, contenant suffisamment de données pour tester l’application. Ce dossier doit être placé à la racine du programme. Pour lancer les tests, il suffit de lancer la méthode Main() de la classe TestModifications.java. Toutes les nouvelles fonctionnalités ajoutées au système sont testées.

1. La liste des projets auxquels une ressource était déjà affectée avant l’exécution courante du système est affichée.
2. Tous les rôles ayant été assignés à un projet spécifique, incluant les rôles assignés avant l’exécution courante **et** les rôles assignés durant l’exécution, sont affichés
3. Une erreur est lancée quand on tente d’assigner une ressource à un projet si ça l’occupait plus de 100 % de son temps. De plus, l’occupation de la ressource est basée sur la priorité du projet.

Alternativement, il est possible de tester manuellement les nouvelles fonctionnalités :

1. Partez le programme, entrez 6 et choisir une ressource pour afficher les projets initialement associés à cette ressource.
2. Partez le programme, entrez 5 et assignez autant de ressources que voulut à un projet. Ensuite, entrez 7 et choisissez ce projet pour afficher tous les rôles lui ayant été assigné.
3. Partir le programme, entrez 5 pour assigner une ressource à un projet et notez la priorité de ce projet. Puis, entrez de nouveau 5 et associez cette même ressource à un autre projet. Si l’occupation[[1]](#footnote-1) de la ressource est plus de 100 % après avoir été assigné à ce nouveau projet, une erreur est lancée. Sinon, l’association est un succès.

1. Projet de priorité H = occupe à 100%, projet de priorité M = occupe à 50%, projet de priorité L = occupe à 25&. [↑](#footnote-ref-1)