Master 1 ALMA

Projet de Vérification et tests

EMF Project

Étudiants :
Coraline MARIE et Vincent RAVENEAU

Intervenant :
Gerson SUNYÉ

13 décembre 2013

Sommaire

1 Introduction

La création d'un logiciel, quel qu'il soit, n'est jamais sûr. Le code source peut contenir des fautes, des bugs ou des défauts, sans même que le programmeur ne les remarque. Fort heureusement, il existe aujourd'hui des méthodes efficaces pour contrer ces failles, et renforcer la fiabilité du code source, comme par exemple les tests unitaires.

Le module *Vérification et Tests* que nous avons étudié en Master Alma à l'Université de Nantes, nous a apporté un ensemble de techniques utilisables sur la majorité des langages informatique. Pour parfaire cet apprentissage, nous avons travaillé sur la restructuration et l'amélioration de l'EMF (Eclipse Modeling Framework), un logiciel libre développé pour et par Eclipse.

Ce rapport présente donc une application directe de ce que nous a appris le module Vérification et tests, au travers de deux grandes étapes de travail. Pour la première, nous avons tout d'abord restructuré une partie du code source de l'EMF en un projet Maven. Puis nous avons évalué les tests déjà présent dans ce code. Pour la seconde étape nous avons amélioré autant que possible la qualité de la classe URI, afin d'augmenter sa fiabilité.

2 Restructuration et évaluation

2.1 Réorganisation avec Maven

L'un des objectifs pédagogiques du projet de vérification et tests, était de nous apprendre à utiliser un nouvel outil : Maven.

2.1.1 Qu'est ce que Maven ?

Maven (ou Apache Maven) est un logiciel libre, qui permet la gestion et la production automatique de projets liés au langage de programmation Java. Il s'agit d'un outil, qui aide à la conception de logiciels à partir du code source, en optimisant les tâches et en garantissant le bon ordre de fabrication.

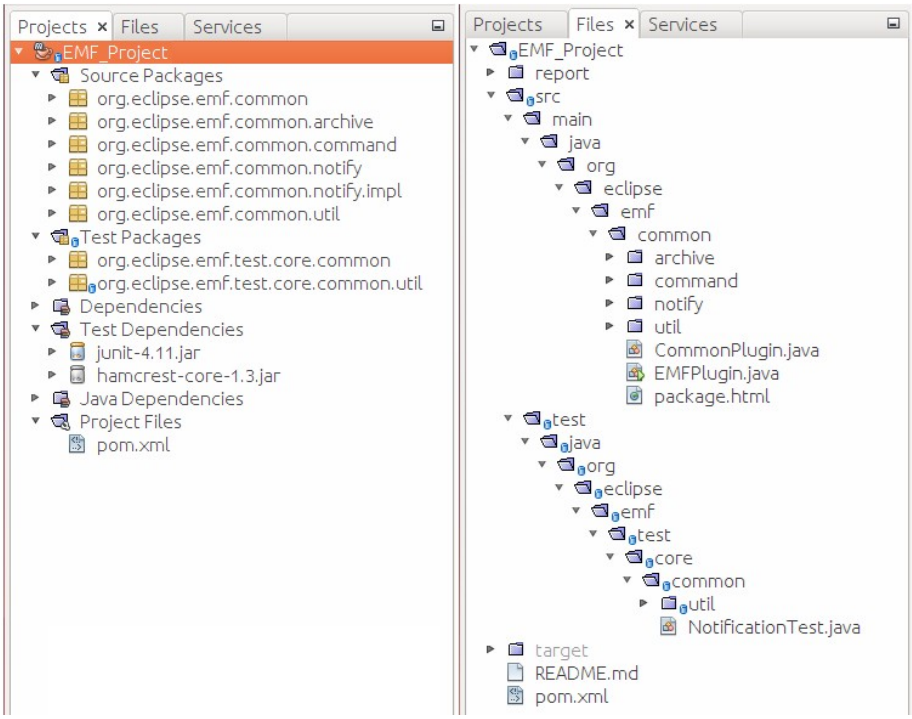
2.1.2 Convertion du code original

Le code source original du projet EMF est disponible sur gitHub. Il est donc téléchargeable en archive, et modifiable par n'importe qui. Cependant, l'archive que nous avons récupérée fournit un code source conditionné pour l'environnement de développement Eclipse, et cet IDE nous a été fortement déconseillé.

Ainsi, pour répondre à la demande du cahier des charges de ne pas utiliser Eclipse, nous avons choisi un nouvel IDE compatible avec Maven pour travailler. Ce choix s'est porté sur l'IDE NetBeans, qui permet de supporter plusieurs langages, et dont la dernière version inclut Maven. Cependant, le code source original devait être au préalable adapté, avant de pouvoir le travailler.

Afin d'optimiser notre travail, nous n'avons pas converti la totalité de l'archive téléchargée, mais juste la partie du code intéressante pour le projet. Malgré la facilité de conversion de l'architecture, cette tâche fût longue et relativement fastidieuse. Certaines erreurs de package et de plugin sont apparues, ce qui nous a pris du temps à résoudre.

Voici l'arborescance obtenue après la conversion du code source original en projet Netbean/Maven :



2.1.3 Bugs et résolution

Pour gérer les bibliothèques et les dépendances d'un projet, Maven utilise un fichier `pom.xml`, qui décrit l'ensemble des besoins de l'application. Lors de la conversion du code source initial vers notre projet Maven, plusieurs erreurs liées à ce fichier sont apparues, car il manquait des dépendances :

- JUnit : framework de test unitaire.
- Core Runtime : framework (de Eclipse RCP).
- EMF ecore et EMF ecore xmi.

2.2 Evaluation des tests

Après avoir lancé les tests, déjà présents dans l'archive initiale, plusieurs problèmes sont apparus. Tout d'abord, il y a eu très peu de couverture du code par les tests (d'après l'IDE Netbeans, moins de 40 % du code est couvert). Ce problème est principalement dû au fait qu'il y ait deux classes de tests, qui possèdent chacune des dépendances que l'on ne peut pas satisfaire. De plus, le logiciel qui permet de tester efficacement les mutations, ne fonctionne pas correctement, et n'exécute pas les bons tests.

3 Amélioration de URI

3.1 La classe URI

3.1.1 Défauts de conception

Après avoir terminé la première partie du projet, c'est à dire réorganiser le projet pour Maven et réparer les erreurs de compatibilités, nous avons chercher les défauts de conceptions de la classe URI.

3.1.2 Améliorations possibles

Pour améliorer la classe URI, nous avons eu plusieurs idées. Malheureusement, nous n'avons pas eu le temps de tester nos hypothèses et de mettre en place nos améliorations.

4 Conclusion

Bien que nous n'ayons pas totalement terminé ce travail, le projet de Vérification et Tests reste un projet d'apprentissage. Nous avons appris de nouvelles notions, comme l'utilisation de Maven et de son système d'architecture. Nous avons également amélioré nos compétences en informatique en travaillant sur le code de d'autres programmeur, et en le retravaillant.

Ce projet nous a permis d'utiliser nos nouvelles connaissances étudiées en *Vérification et Tests*, mais pas seulement. Nous nous sommes servis pour ce projet de d'autres notions vues dans d'autres modules d'enseignement, comme par exemple *Concepts et Outils de Développement*, ou *Génie Logiciel*.