OPL – Code Rewinder

*Le réparateur automatique de bugs par agrégation de code multi-version*

Code Rewinder est un réparateur automatique de bugs découpé en trois principales parties. Dans un premier temps, il télécharge les n dernières versions du projet sur GitHub. Dans un second temps, il construit, grâce à ces versions, une nouvelle version qui contient toutes les versions des fonctions. Et pour finir, il exécute les tests unitaires tout en faisant varier les versions des méthodes jusqu’à trouver le plus de tests « successful » possibles.

WATTEBLED Étienne

[Choisir la date]

OPL – Code Rewinder

Le réparateur automatique de bugs par agrégation de code multi-version

Table des matières

[Introduction 2](#_Toc469431314)

[Travail technique 3](#_Toc469431315)

[Améliorations préalables 3](#_Toc469431316)

[Analyses et corrections 4](#_Toc469431317)

[Exécution de Code Rewinder sur demoproject 4](#_Toc469431318)

[Exécution de Code Rewinder sur jsoup 5](#_Toc469431319)

# Introduction

Code Rewinder est un réparateur automatique de bugs utilisant les n derniers commits de SVN pour parvenir à ses fins.

Il fonctionne principalement en trois temps :

* Avant tout, il doit exécuter un batch permettant de télécharger les n dernières versions du projet à réparer sur GitHub.
* À l’aide de ces versions, il construit en utilisant une librairie appelée Spoon, une nouvelle version comprenant toutes les versions des méthodes. Un switch sur un attribut static de la classe permet de passer une méthode d’une version à l’autre.
* Et il exécute ensuite les tests unitaires tout en faisant varier les versions des méthodes jusqu’à obtenir le maximum de tests « successful » possibles.

Avant tout, quelques améliorations ont été apportées à Code Rewinder.   
Par la suite, Code Rewinder a été exécuté pour la première fois sur un projet concret et des corrections ont été apportées. Des tests unitaires ont alors été mis en place afin d’éviter qu’il y ait une quelconque régression dans un futur plus ou moins éloigné.

# Travail technique

## Améliorations préalables

Des améliorations et configurations ont été apportées à Code Rewinder, même si celles-ci n’étaient pas toutes importantes :

1 – La plupart des utilisateurs dispose d’un OS Windows. Malheureusement, le batch permettant de télécharger les n dernières versions d’un projet github était conçu pour Linux. Un batch Windows a donc été mis en place afin de satisfaire un maximum d’utilisateurs.

2 – Le réparateur était divisé en deux parties, une partie concernant le téléchargement des versions et la transformation, et une autre partie permettant d’exécuter les tests unitaires et donc, de réparer. Les deux parties ont donc été liées afin de simplifier l’exécution de Code Rewinder.  
  
3 – Lors du chargement des classes avec Reflections, le nom du package était autrefois écrit en dur dans l’application. Un argument -packages a donc été ajouté et celui-ci permet de lister les packages du projet à réparer dans Code Rewinder (chaque package doit être séparé par un point-virgule, comme le classpath). Il est même possible même de préciser le nom de la classe avec le package et, indiquer le nom du package en entier n’est pas obligatoire (si le package des classes contient un de vos mots alors la classe est prise en compte, qu’il s’agisse d’une classe de test ou d’une classe du projet).

4 – L’utilisation de StringBuilder au-delà de la concaténation de deux chaînes de caractères a été favorisée afin de gagner en performances.

## Analyses et corrections

### Exécution de Code Rewinder sur demoproject

Afin de faire évoluer Code Rewinder Progressivement, ce dernier n’a pas été exécuté de suite sur un gros projet. Il a tout d’abord été exécuté sur un petit « projet » appelé demoproject (avec trois classes et une seule classe de tests qui ne contient qu’un seul test toujours vrai) qui était fourni dans la version bêta-bêta. Le but de demoproject était vraiment d’avoir une base un minimum fonctionnelle avant de se lancer dans l’exécution de Code Rewinder sur un véritable projet.

Code Rewinder utilise une librairie Java appelée Reflections qui permet entre-autres d’accéder et de modifier la valeur des attributs static des classes (et donc, de changer la version d’une méthode). Cependant, la dépendance n’était pas présente dans le POM Maven : elle a été ajoutée.

Un autre problème fit par la suite son apparition, en réalité, Reflections ne parvenait pas à accéder aux attributs des classes car ces derniers étaient « private ». Ils ont donc été changés en « public ».  
Cependant, ces attributs étaient déclarés « final », le changement de valeur n’était donc pas autorisé par Java. Le mot clé « final » a donc dû être supprimé afin que Code Rewinder puisse changer la version des méthodes.

Malgré ces corrections sur les attributs, une autre exception s’est déclenchée durant l’analyse. Les attributs étaient de type Integer, alors que, lors de la modification des valeurs, il était question de type primitif « int ». Cette erreur a été corrigé en créant à chaque fois un nouvel Integer et en passant le type primitif au constructeur.

Le dernier problème avec demoproject était que, les classes n’étaient pas automatiquement compilées et envoyées dans le target du projet Code Rewinder. Par conséquent, Reflections ne parvenaient pas à retrouver (charger) les classes. La compilation et le déplacement des fichiers .class ont été effectués grâce à l’argument « --compile » de Spoon.

Grâce à ces modifications, le projet demoprojet a par la suite été fonctionnel pour la première fois et fonctionnait correctement sur Code Rewinder.

### Exécution de Code Rewinder sur jsoup

Code Rewinder a, par la suite, été testé sur Jsoup, qui est quant à lui un projet de taille réelle avec lequel de multiples erreurs sont survenus durant l’analyse.

La première anomalie a été le fait qu’un paramètre d’une méthode a changé de nom. Par conséquent, Code Rewinder ne parvenait pas à faire le lien entre l’ancien nom de variable et le nouveau. Ce problème a été corrigé.

C’est alors qu’un problème beaucoup plus grave est apparu. En réalité, dans un cas très particulier, l’héritage provoquait une boucle infinie qui générait un StackOverflowError. Le cas particulier est le suivant :  
- Deux classes : A et B  
- B qui hérite de A :  
- Une méthode m commune à A et à B  
- La méthode m de la classe mère A fait un appel à super pour appeler la méthode m de B.

public class A {

public void m() {

…

}

}

public class B extends A { public void m() {

…

super.m() ;

…

}

}

Ces classes, après transformation de Code Rewinder sur 2 commits deviennent alors :

public class A {

public void m() {

…

}

public Integer m\_version = 0 ;

public Integer m\_version\_max = 1 ;

}

public class B extends A {

public void m() {

switch(m\_version) {

case 0 : m\_0() ;

case 1 : m\_1() ;

}

}

public Integer m\_version = 0 ;

public Integer m\_version\_max = 1 ;

}