Rapport de stage

Remédiation de masse dans un environnement d'apprentissage

Alexandre Carpentier

Année 2014–2015

Stage de deuxième année réalisé au LORIA

en vue de la validation de la deuxième année d'études

Maître de stage : Martin Quinson

Encadrant universitaire : Gérald Oster

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Carpentier, Alexandre

Élève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31313703

Année universitaire : 2014–2015

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Remédiation de masse dans l'environnement PLM dédié à l'apprentissage de la programmation

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Vandœuvre-lès-Nancy, le 20 août 2015

Signature :

Rapport de stage

Remédiation de masse dans un environnement d'apprentissage

Alexandre Carpentier

Année 2014–2015

Stage de deuxième année réalisé au LORIA

en vue de la validation de la deuxième année d'études

Alexandre Carpentier
9, square de Liège
54500, VANDŒUVRE-LÈS-NANCY
+33 (0)6 59 07 39 97
alexandre.carpentier@telecomnancy.eu

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

LORIA
Campus Scientifique
54506, VANDŒUVRE-LÈS-NANCY
+33 (0)3 83 58 17 50

Laboratoire lorrain de recherche en informatique et ses applications

Maître de stage : Martin Quinson

Encadrant universitaire : Gérald Oster

Remerciements

Je tiens à remercier toutes les personnes qui ont pu m'aider à aboutir au résultat obtenu au jour de l'écriture de ce rapport.

Tout d'abord, j'adresse mes remerciements à mes professeurs et maîtres de stage, MM. Martin Quinson et Gérald Oster, pour leur soutien et leur écoute tout au long du stage, ainsi que pour la confiance qu'ils m'ont accordée pour la réalisation du projet.

Je tiens également à remercier à M. Matthieu Nicolas, pour toute l'aide qu'il a pu m'apporter pour comprendre les méandres du code de la PLM, mais aussi pour tous les bons moments qu'on a pu passer lors de ces dix semaines intenses.

Enfin, je remercie toutes les personnes qui m'ont aidé à écrire et à relire ce rapport de stage.

Table des matières

Remerciements **v**

Table des matières **vii**

Introduction **1**

1 Présentation de l'entreprise et du projet **3**

1.1 L'entreprise : le LORIA 3

1.2 La *PLM* et ses satellites 3

1.2.1 Le noyau : la *PLM* 3

1.2.2 La nouvelle version : *webPLM* 4

1.2.3 La base de données : *PLM-data* 4

1.2.4 La bibliothèque de parcours des données : *PLM-reaper* 4

1.3 Le projet 4

1.3.1 Problématique 4

1.3.2 Méthode de travail 4

2 Parcours et exécution du code des élèves **7**

2.1 Les traces des élèves 7

2.2 Exécution du code récupéré 8

2.3 Edition et traitement de statistiques 9

3 De l'erreur classique à l'indice **11**

3.1 La méthode de comparaison des erreurs classiques dans la *PLM* 11

3.2 L'ajout des indices dans les exercices 12

4 Les indices dans la version web **14**

4.1 Mise à jour de la version de *PLM* utilisée dans *webPLM* 14

4.2 Le mécanisme de retour utilisateur 14

Conclusion **15**

Liste des illustrations **17**

Liste des tableaux **19**

Glossaire **21**

Résumé **23**

Abstract **23**

Introduction

La *"Programmer's Learning Machine"* (abrégée *PLM* dans la suite de ce rapport) est un outil d'apprentissage de la programmation puissant. Il permet à un débutant d'apprendre les bases de la programmation dans divers langages (Java, Scala, Python). Il est notamment utilisé lors des deux premières semaines de la rentrée à TELECOM Nancy, pour que les élèves venus de classe préparatoire aux grandes écoles apprennent rapidement à programmer. Le nombre de ces élèves étant proche de la centaine, les deux professeurs qui encadrent ce module peuvent être débordés par les demandes.

Le but de mon stage est d'assister ces professeurs lors de l'apprentissage, en fournissant aux élèves des suggestions pour débloquer certaines situations particulières. Depuis un an, la *PLM* demande aux élèves s'ils veulent partager leur code. J'ai donc étudié cette "base de données" anonyme afin de déterminer avec quelle fréquence certains motifs pouvaient apparaître. L'ensemble de ces situations particulières mènent à un résultat que l'on appelle "remédiation de masse" : elle permet en effet de mener les élèves qui ont fait un certain type d'erreur vers la solution (c'est la "remédiation"), sans que cela ne soit totalement personnalisé — tout le monde peut y accéder.

Le projet s'est ainsi déroulé en suivant trois grandes étapes. Tout d'abord, il a fallu parcourir les traces des élèves et ré-exécuter leurs codes, pour obtenir des statistiques et déterminer les erreurs les plus communes. J'ai ensuite ajouté le code permettant d'afficher des suggestions tirées des erreurs classiques dans la *PLM*. Finalement, nous avons intégré le code de la *PLM* à sa version web.

La structure de ce rapport suivra donc ce cheminement après une présentation plus détaillée du contexte du stage et une description précise des outils déjà présents au début de ce stage.

1 Présentation de l'entreprise et du projet

1.1 L'entreprise : le LORIA

Le LORIA (Laboraoire Lorrain de Recherche en Informatique et ses Applications) est une Unité Mixte de Recherche. En son sein, plusieurs équipes sont réparties entre cinq départements, qui couvrent un large spectre du domaine de la recherche fondamentale et appliquée en informatique. Le LORIA se situe à Vandœuvre-lès-Nancy, à proximité de TELECOM Nancy. Le LORIA compte pas moins de 450 employés, ce qui en fait l'un des plus grands laboratoires de recherche en sciences informatiques de Lorraine.

J'ai été accueilli dans l'équipe VERIDIS, dont la thématique principale est la conception de systèmes distribués et la vérification des systèmes. Cette équipe fait partie du département d'étude des méthodes formelles. Cependant, la *PLM* n'appartenant pas à proprement parler à cette équipe, j'en étais quelque peu détaché. L'organigramme de l'entreprise de l'année 2014 est présent en annexe et dénote de l'homogénéité qui existe entre l'INRIA, le LORIA et le CNRS.

1.2 La *PLM* et ses satellites

1.2.1 Le noyau : la *PLM*

Figure 1.1 – Ecran d'accueil de la *PLM*

FIGURE 1.2 – Ecran d'exercice de la *PLM* avec erreur et indice

1.2.2 La nouvelle version : *webPLM*

FIGURE 1.3 – Ecran d'accueil de *webPLM*

1.2.3 La base de données : *PLM-data*

1.2.4 La bibliothèque de parcours des données : *PLM-reaper*

1.3 Le projet

1.3.1 Problématique

1.3.2 Méthode de travail

FIGURE 1.4 – Ecran d'exercice de *webPLM*

2 Parcours et exécution du code des élèves

L'étude des traces des élèves a été la première grande étape du projet. La moitié du temps du stage a été consacré à cette partie. Les traces des élèves sont gardées dans un dépôt Git sur GitHub et c'est ce que j'ai dû étudier au cours de mon PIDR, puis au début du stage.

2.1 Les traces des élèves

Les traces des élèves sont représentées dans ce qui s'apparente à une base de données sous Git. Chaque code de l'élève est versionné, c'est-à-dire qu'à chaque exécution de son code, il "commite" son code sous *PLM-data*.

La forme est simple : un élève est représenté par une branche. Sur chacune de ces branches, on retrouve les exercices qui ont été ouverts ou exécutés par l'élève en question, sous la forme d'un commit contenant un fichier de code, un fichier de correction, un fichier de mission et un fichier d'erreur, ainsi que, le cas échéant, un fichier de réussite de l'exercice. Le titre du commit permet quant à lui d'obtenir plusieurs informations concernant notamment la réussite ou non d'un exercice, le système d'exploitation utilisé, la version de *PLM*...

FIGURE 2.1 – Extrait de la vue sur GitHub d'une branche de *PLM-data*

Lors de l'étude des traces des élèves, il a fallu, à l'aide d'un parseur de dépôt Git, vérifier si l'élève avait bien obtenu une erreur d'exécution et pas une réussite ou une erreur de compilation. Pour cela, on regarde le titre du commit, puisqu'il contient toutes les informations pour déterminer l'état d'un exercice.

La méthode principale récupère le code de l'élève. Pour cela, j'ai parcouru le dépôt Git et en ai observé tous les titres de commit pour en extraire ceux qui mènent à une erreur. Un affichage avec une barre de chargement permet de voir en continu l'exécution du programme.

Pendant ce parcours, l'étude des commits est ainsi faite : on vérifie que le code a été exécuté, qu'il a bien mené à une erreur, puis on en récupère le langage de programmation, le nom de la leçon et de l'exercice, que l'on modifie si l'exercice n'a plus le même nom.

Avant d'exécuter le code de l'élève ainsi obtenu, il nous faut encore l'épurer en suivant certains critères :

- si, lors du parcours de la branche et dans l'exercice précis, un code est identique à un autre, on passe au code suivant ;
- si le code a déjà été étudié lors d'une exécution antérieure, c'est-à-dire qu'il est présent dans le cache, il n'est pas nécessaire de regarder à nouveau ce code.

2.2 Exécution du code récupéré

Le code est alors exécuté ; pour cela, on passe en paramètres de la méthode de ré-exécution toutes les données nécessaires sur l'exercice, la leçon, le code, le commit. On lui passe également un "Game", que l'on va préparer en précisant divers paramètres comme l'exercice courant et le code écrit.

Après avoir réglé l'objet "Game", une partie du code sert à lancer l'exécution du code, tout en vérifiant que l'on ne dépasse pas la durée maximale fixée dans les paramètres (à titre d'information, cette durée a été fixée à 7,5 secondes).

Il a été cherché un moyen de calculer plus efficacement cette durée maximale, notamment en cherchant à déterminer le temps d'exécution de la solution. Cependant, sur certains types d'exercices, il est impossible de réaliser un tel calcul, comme par exemple, les exercices de tri qui ont un monde de départ aléatoire (ces exercices ont d'ailleurs été retirés de la liste obtenue avant l'exécution).

Une fois l'exécution terminée, on compare le monde final obtenu au monde correction et on récupère dans un fichier texte la différence entre eux. C'est cette différence qui donne l'erreur qui sera par la suite étudiée.

Figure 2.2 – Contenu du dossier d'erreurs d'un exercice

Ces fichiers texte ne sont pas abandonnés en l'état, ils sont préalablement classés par exercice

(le nom de l'exercice comprenant le nom de la leçon associée) et ont pour nom un texte de la forme : [ID_de_la_branche]_[ID_du_commit].log, ce qui permet de très facilement retrouver le code associé à une erreur particulière directement sur le dépôt GitHub.

2.3 Edition et traitement de statistiques

Une fois ces fichiers tous obtenus, il faut traiter la masse de données obtenues. Pour cela, il a été choisi lors du PIDR de créer un fichier tableur pour rendre le parcours et l'analyse de ces données plus aisé. Ainsi, j'ai utilisé une structure de table de hachage double :

- la clé représente l'identifiant de l'exercice ;
- le contenu est une table de hachage dont :
  - la clé représente le contenu du fichier d'erreur ;
  - le contenu est un vecteur qui contient le nom du fichier de l'erreur.

Ainsi, le répertoire contenant les fichiers d'erreur est scanné et toute sa substance est résumée dans cette structure complexe. L'étape suivante est le traitement de ces données.

Pour cela, j'ai choisi d'utiliser la librairie Apache POI (sous licence Apache) pour permettre à l'utilisateur de choisir l'extension de sortie du tableur. Il est préféré une sortie au format *.xlsx*, des dernières versions d'Excel, puisqu'elle est plus permissive sur le contenu des cellules du tableur, principalement sur la taille de ce contenu (plus de 32 000 caractères).

Ainsi, la première colonne donne le nom de l'exercice, la seconde le texte de l'erreur, la troisième donne le nombre d'erreurs identiques, la quatrième donne le nombre de session par erreur identique.

Une fois ce tableur obtenu, il faut passer à son analyse. Pour cela, des calculs sont effectués pour déterminer l'importance des erreurs. J'ai pu par exemple déterminer le nombre d'erreurs moyen par session, ce qui a pu me mener à un moyen de calculer l'urgence d'un exercice.

Figure 2.3 – Capture d'écran avec légende du tableur final

Pour déterminer l'urgence du traitement d'un exercice, j'ai réalisé un calcul sur des seuils au niveau du nombre d'erreurs identiques, ainsi que sur le nombre moyen d'erreur par session. Les seuils sur les nombres d'erreurs et de la moyenne d'erreur par élève ont été calculés selon un

rapport exponentiel sur leur maximum respectif. Le tableau ci-dessous décrit la valeur associée à chaque palier d'erreur ou de moyenne par élève.

Nombre d'erreurs	**Valeur associée**	**Moyenne d'élèves**	**Valeur associée**
$x \geq 55$	4	$y \geq 7.1$	1
$55 > x \geq 25$	2	$7.1 > y \geq 5$	0
$25 > x \geq 16$	1	$5 > y \geq 4.15$	-1
$16 > x \geq 0$	0	$4.15 > y \geq 0$	-2

TABLE 2.1 – Tableau des seuils d'erreur et de moyenne d'erreur par élève

En corrélant le nombre d'erreurs identiques avec cette valeur moyenne grâce à la somme des valeurs précédentes, j'ai pu ainsi déterminer un facteur d'importance de l'erreur qui m'a permis de classer les exercices les plus urgents à traiter. L'importance varie entre 0 et 5 : si la somme est inférieure à 0, elle est ramenée à 0.

3 De l'erreur classique à l'indice

La partie qui suit l'obtention du tableau des erreurs classiques est celui de l'analyse des valeurs obtenues. Cette étape s'est divisée en deux parties : j'ai tout d'abord augmenté le code de la *PLM*, puis j'ai ajouté les fichiers d'indices et le code des erreurs associées.

3.1 La méthode de comparaison des erreurs classiques dans la *PLM*

Pour savoir à quel moment un élève a réalisé ce que j'ai appelé jusqu'ici une erreur classique, il a fallu les intégrer dans la *PLM*. Cela a impliqué la modification des modules de vérification, des changements dans les variables propres aux exercices et au processus d'exécution, ainsi que dans la mise en lace de la structure des exercices en général.

L'idée est simple : on ajoute le code de l'erreur classique souhaitée, on teste le code de l'élève et si l'élève n'aboutit pas à la solution attendue, on compare sa solution aux mondes des erreurs classiques. Si le monde courant final correspond à l'un des mondes des erreurs classiques, alors le programme donne le contenu du fichier d'indice correspondant à l'élève.

Pour entrer dans des détails plus techniques, la majorité du code modifié a été dans l'implémentation des exercices. Cela concerne principalement deux classes, "Exercise" et "ExerciseTemplated", qui possèdent les méthodes *setupWorlds* et *check* pour la première, et *setup* et *computeError* pour la seconde.

Au lancement de la *PLM*, après le choix de la leçon par l'élève, la méthode *setup* est appelée. Elle permet de récupérer tous les fichiers d'erreur par exercice et de mettre le chemin relatif dans une ArrayList de chaîne de caractères. On passe ensuite cette ArrayList à la méthode *computeError*. Cette méthode permet de récupérer et d'exécuter dans un Thread à part le code contenu dans les fichiers d'erreurs obtenus précédemment.

Du côté des exercices, la méthode *setupWorlds* a été améliorée, pour permettre la prise en compte des fichiers d'erreur. Telle que je l'ai écrite, elle ne limite pas le nombre de ces fichiers. La structure retenue est celle qui était déjà utilisée pour stocker les mondes courant, initial et solution : il s'agit de vecteurs de *World*. Cependant, étant donné que le code associé à ces trois mondes est unique, et que les erreurs peuvent être multiples, la structure est un vecteur de vecteurs de *World*.

Finalement, la méthode *check* est appelée une fois que l'exécution est terminée. On y initialise les variables du processus d'exécution qui correspondent aux erreurs communes : le texte de l'indice commence en étant une chaîne de caractères vide et l'identifiant de l'indice est -1. Ces valeurs de départ permettent de savoir si une erreur classique a été trouvée. On vérifie ensuite

tous les mondes qui constituent l'exercice sur deux points : le monde solution et les mondes erreur. La démarche est la suivante :

1. si le monde obtenu par l'élève correspond au monde de la correction, tout va bien et l'élève peut passer à l'exercice suivant l'esprit tranquille ;
2. sinon, on vérifie tous les mondes erreurs les uns après les autres jusqu'à ce qu'il n'y en ait plus ou lorsque qu'une correspondance est trouvée :
   — s'il y a une correspondance, on modifie la valeur du texte de l'indice par celle du contenu du fichier d'indice et la valeur de l'indice par celle de l'erreur ;
   — sinon, seul le texte de l'erreur (la différence entre le monde final et le monde solution) s'affiche à l'écran de l'élève. Il faut noter que ce texte s'affiche dans tous les cas si l'élève obtient une erreur.

De nombreux fichiers ont été modifiés, mais il ne s'agit que de modifications mineures et il ne serait pas pertinent d'aborder cela ici. Si cependant ces modifications intéressent le lecteur, je vous invite à vus rendre sur GitHub, dans les dépôts de l'utilisateur nommé BuggleInc, dépôt *PLM*[1].

3.2 L'ajout des indices dans les exercices

Une fois le moteur terminé, il a été nécessaire d'étudier les résultats contenus dans le tableur avec les erreurs. Pour cela, le critère d'importance a permis de voir rapidement les exercices à reprendre.

La démarche d'étude des erreurs classiques a été simple :
— j'ai pris les erreurs dans l'ordre décroissant d'importance (en partant de 5) ;
— en utilisant l'IDE Eclipse pour faire des recherches dans les fichiers, j'ai déterminé quels fichiers possédaient l'erreur sélectionnée ;
— je récupère ensuite l'identifiant de commit dans le nom des fichiers ;
— sur GitHub, je me déplace dans le dépôt *PLM-data*, et je copie l'identifiant du commit ;
— le code est ensuite récupéré dans le fichier correspondant ;
— il est finalement testé pour bien vérifier qu'il mène à la bonne erreur.
Une fois le code testé et récupéré, il est passé dans le projet, en suivant une procédure de traduction : en effet, la plupart du temps, le code élève est écrit en Python ou en Scala, mais très rarement en Java. Le code qu'on a alors obtenu est traduit, puis modifié, car parfois, ce que l'élève a écrit ne s'exécute pas et il n'est pas nécessaire de laisser une partie de code comme celle-là dans le fichier d'erreur classique.

Les fichiers d'erreurs classiques doivent avoir une forme particulière. Tout d'abord, il faut bien évidemment les mettre dans le répertoire qui contient l'exercice en question. Le point suivant concerne leur nom : il doit avoir comme forme [nom_exercice]CommonErr[0-9]*.java. Il faut de plus que chaque fichier précédant un nombre existe : il ne peut y avoir de fichier CommonErr2 sans un fichier CommonErr1 et CommonErr0.

Quant au contenu de ces fichiers, il faut qu'il y ait les mêmes méthodes que le fichier Entity correspondant à l'exercice. Ce fichier Entity est celui qui donne la correction de l'exercice dont le nom est précisé auparavant.

L'étape suivante est la rédaction de l'indice. Il est impératif de rédiger son indice en anglais. De

1. https://github.com/BuggleInc/PLM

plus, le nom du fichier d'indice pour une erreur donnée est la même, à l'extension près : il faut que le fichier soit en HTML dont l'extension est .html. L'indice correspondant à une erreur doit cependant avoir le même numéro écrit après "CommonErr". L'indice peut être écrit sur plusieurs lignes, ce qui permet une certaine liberté dans sa rédaction.

La dernière étape est leur traduction. Pour cela, l'outil po4a est utilisé. En modifiant le fichier po4a.conf, c'est-à-dire en lui ajoutant tous les nouveaux fichiers à traduire, puis en passant par l'outil poedit, j'ai pu directement récupérer les contenu des fichiers html en anglais et les traduire en français.

4 Les indices dans la version web

4.1 Mise à jour de la version de *PLM* utilisée dans *webPLM*

4.2 Le mécanisme de retour utilisateur

Conclusion

Liste des illustrations

1.1 Ecran d'accueil de la *PLM* 3
1.2 Ecran d'exercice de la *PLM* avec erreur et indice 4
1.3 Ecran d'accueil de *webPLM* 4
1.4 Ecran d'exercice de *webPLM* 5

2.1 Extrait de la vue sur GitHub d'une branche de *PLM-data* 7
2.2 Contenu du dossier d'erreurs d'un exercice 8
2.3 Capture d'écran avec légende du tableur final 9

Liste des tableaux

2.1 Tableau des seuils d'erreur et de moyenne d'erreur par élève 10

Glossaire

Résumé

Mots-clés :

Abstract

Keywords :