Code Inspection Report

Anti-Spam Configuration Software Development Project

BSc/MSc in [LEI | LIGE | METI]
Academic Year 2017/2018 - 1º Semester
Software Engineering I

Group 21
73263, André Godinho, EIC1
73324, Paulo Pina, EIC1
72897, Rodrigo Agostinho, EIC1
73289, Susana Gomes, EIC1

ISCTE-IUL, Instituto Universitário de Lisboa
1649-026 Lisbon
Portugal

November 25th 2017

Table of Contents

Introduction 3
Code inspection – Functions 3
Code inspection checklist 3
Found defects 3
Corrective measures 4
Conclusions of the inspection process 4
Code inspection – Interface 4
Code inspection checklist 4
Found defects 5
Corrective measures 5
Conclusions of the inspection process 5

Introduction

The system consists in an anti-spam filtering using jMetal Optimization Framework where rules are grouped according to their type and each rule is checked (true/false) for each email message and has associated a positive (spam) or negative (legitimate) weight. If messages total (sum of) score is higher than a pre-defined threshold they are classified as SPAM, otherwise classified as HAM.

Code inspection – Functions

The component being inspected, Functions, contains the methods that will be used in both Automatic Configuration and in Manual Configuration. This methods center around the reading of the files: rules.cf, ham.log, spam.log. The methods are used to count the number of lines (number of rules), returning an array with the rules, returning an array with the weights, write in file the new weights, converting the content of a file to an array, evaluate the given solution, choose the best solution, return the solution of the given line and returning the false positives and the false negatives of the given line.

Meeting date: *Meeting duration:* *Moderator:* *Producer:* *Inspector:* *Recorder:*	*22/12/2017* *60 minutes* *Rodrigo Agostinho* *Todos os elementos do grupo* *Susana Gomes* *Rodrigo Agostinho*
Component name (Package/Class/Method):	*AntiSpamFilter/Functions*
Component was compiled:	*Yes*
Component was executed:	*Yes*
Component was tested without errors:	*Yes*
Testing coverage achieved:	*78.1%*

Code inspection checklist

The checklist given on Blackboard was used to assess if the class Functions met all the requirements. Overall, the class checked every point that applied to it. The most important checklists to assess this class were: Variable, Attribute, and Constant Declaration Defects (VC), Method Definition Defects (FD), Control Flow Defects (CF), Input-Output Defects (IO), Comment Defects (CM), Layout and Packaging Defects (LP), Modularity Defects (MO), Performance Defects (PE).

Found defects

Found defect Id	Package, Class, Method, Line	Defect category	Description
1	AntiSpamFilter/Functions-write_weights(String path, ArrayList<Double> Solution)	IO	In the case of the Automatic Configuration, instead of overwriting the rules in AntiSpamConfigurationForBalancedProfessionalAndLeisureMailbox/rules.cf, it was just adding the new weights. So, in case of different rules in the files, it was

			keeping the ones in AntiSpamConfigurationForBalancedProfessionalAndLeisureMailbox/rules.cf, instead of writing there the rules inside of user's rules.cf. In the case of the Manual Configuration, there wasn't any problem because it reads and writes in the same file (user's rules.cf).

Corrective measures

Id: 1 – André solved the defect with Paulo's help while this report was being written. Now, the method is called write_weights_and_rules().

Conclusions of the inspection process

This component was built from scratch, with the checklist in mind, so there weren't any defects found, and no corrections needed. The component integrates well with the software to be delivered, being needed to its functional purpose.
Since the component was mostly written by the Quality Manager, the component was assessed by the other members of the group to assure that the quality management wasn't compromised.

Code inspection – Interface

The component being inspected, Interface, contains the GUI interface that will be presented to the user. This interface contains the path input to file rules.cf, ham.log, and spam.log that should be introduced by the user, the manual implementation that shows each rule and correspondent weight such as the button to test this solution and to save it in rules.cf and also presents the overall number of false negatives and false positives associated to the solution, the automatic implementation that generates a list of rules with correspondent weights that will be shown in the interface and also presents the overall number of false negatives and false positives associated to the solution.

Meeting date:	*22/12/2017*
Meeting duration:	*60 minutes*
Moderator:	*Rodrigo Agostinho*
Producer:	*Todos os elementos do grupo*
Inspector:	*Susana Gomes*
Recorder:	*Rodrigo Agostinho*
Component name (Package/Class/Method):	*AntiSpamFilter/Interface*
Component was compiled:	*Yes*
Component was executed:	*Yes*
Component was tested without errors:	*Yes*
Testing coverage achieved:	*70.6%*

Code inspection checklist

The checklist given on Blackboard was used to assess if the class Interface met all the requirements. Overall, the class checked every point that applied to it. The most important checklists to assess this class were: Variable, Attribute, and Constant Declaration Defects (VC), Method Definition Defects (FD), Comparison/Relational Defects (CR), Control Flow Defects (CF), Input-Output Defects (IO), Module Interface Defects (MI), Comment Defects (CM), Layout and Packaging Defects (LP), Modularity Defects (MO), Performance Defects (PE).

Found defects

Found defect Id	Package, Class, Method, Line	Defect category	Description
1	AntiSpamFilter/Interface	LP	The class is more than 600 lines long, although, after some ponderation, we chose not to change it. We think this won't affect the understanding and performance of the component.

Corrective measures

There weren't any corrective measures since the team found that the found defect wouldn't affect the understanding or performance of the component.

Conclusions of the inspection process

This component was built from scratch, with the checklist in mind. The component integrates well with the software to be delivered, being needed to its functional purpose, and being friendly to the user, protecting the system and inputs.
The component was assessed by the all the members of the group.