|  |  |
| --- | --- |
|  | *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 Id ...  69515,Miguel Bento,Lei-PL  72756, Daniel Fernando, Lei-PL  73304, Joao Nuno, EIC  73384, Ricardo Lopes, EIC  ISCTE-IUL, Instituto Universitário de Lisboa  1649-026 Lisbon  Portugal  November 25th 2017 |

**Table of Contents**

[Introduction 3](#_Toc498465002)

[Code inspection – Name of the component being inspected 3](#_Toc498465003)

[Code inspection checklist 3](#_Toc498465004)

[Found defects](#_Toc498465005) 6

[Corrective measures](#_Toc498465006) 6

[Conclusions of the inspection process](#_Toc498465007) 6

# [Introduction](#__RefHeading___Toc2530_1503482439)

*O software desenvolvido pelo nosso grupo (31) foi feito com o objectivo de optimizar o filtro Anti-Spam de uma caixa de correio Profissional, filtro este que é responsavel pela classificação das mensagens recebidas, que ou são spam ou ham. Para isso usamos o algoritmo NSGA-II que corre na framework jMetal do Java, que é a linguagem que vamos usar para desenvolver este software. O objectivo do software é conseguir gerar os pesos ideais para cada regra, o que irá criar uma configuração ideal para uma caixa de correio profissional, em que o objectivo é ter o menor numero possivel de falsos negativos e positivos.*

# Code inspection – Name of the component being inspected

*Description of the software component being inspected*

|  |  |
| --- | --- |
| *Meeting date:*  *Meeting duration:*  *Moderator:*  *Producer:*  *Inspector:*  *Recorder:* | *21/12/2017*  *45 minutes* |
| *Component name (Package/Class/Method):* | *AntiSpamFilter* |
| *Component was compiled:* | *yes* |
| *Component was executed:* | *yes* |
| *Component was tested without errors:* | *yes* |
| *Testing coverage achieved:* | *81%* |

# Code inspection checklist

1. Variable, Attribute, and Constant Declaration Defects (VC)

☑ Are there descriptive variable and constant names used in accord with naming

conventions?

❑ Are there variables or attributes with confusingly similar names?

☑ Is every variable and attribute correctly typed?

☑ Is every variable and attribute properly initialized?

❑ Could any non-local variable be made local?

☑ Are all for-loop control variables declared in the loop header?

❑ Are there literal constants that should be named constants?

❑ Are there variables or attributes that should be constants?

❑ Are there attributes that should be variables?

☑ Do all attributes have appropriate access modifiers (private, protected, public)?

❑ Are there static attributes that should be non-static or vice-versa?

2. Method Definition Defects (FD)

❑ Are descriptive method names used in accord with naming conventions?

☑ Is every method parameter value checked before being used?

☑ For every method: Does it return the correct value at every method return point?

☑ Do all methods have appropriate access modifiers (private, protected, public)?

❑ Are there static methods that should be non-static or vice-versa?

3. Class Definition Defects (FD)

☑ Do all class have appropriate constructors and destructors?

❑ Do any subclasses have common members that should be in the superclass?

❑ Can the class inheritance hierarchy be simplified?

4. Data Reference Defects (DR)

☑ For every array reference: Is each subscript value within the defined bounds?

☑ For every object or array reference: Is the value certain to be non-null?

5. Computations/Relational Defects (CR)

☑ Are there any computations with mixed data types?

❑ Is overflow or underflow possible during a computation?

☑ For each expression with more than one operator: Are the assumptions about order

of evaluation and precedence correct?

☑ Are parentheses used to avoid ambiguity?

6. Comparison/relational Defects (CR)

☑ For every Boolean test: Is the correct condition checked?

☑ Are the comparison operators correct?

❑ Has each Boolean expression been simplified by driving negations inward?

☑ Is each Boolean expression correct?

❑ Are there improper and unnoticed side-effects of a comparison?

❑ Has an “&” inadvertently been interchanged with a “&&” or a “|” for a “||”?

7. Control Flow Defects (CF)

☑ For each loop: Is the best choice of looping constructs used?

☑ Will all loops terminate?

❑ When there are multiple exits from a loop, is each exit necessary and handled

properly?

☑ Does each switch statement have a default case?

☑ Are missing switch case break statements correct and marked with a comment?

☑ Do named break statements send control to the right place?

❑ Is the nesting of loops and branches too deep, and is it correct?

❑ Can any nested if statements be converted into a switch statement?

❑ Are null bodied control structures correct and marked with braces or comments?

☑ Are all exceptions handled appropriately?

☑ Does every method terminate?

8. Input-Output Defects (IO)

☑ Have all files been opened before use?

☑ Are the attributes of the input object consistent with the use of the file?

☑ Have all files been closed after use?

❑ Are there spelling or grammatical errors in any text printed or displayed?

☑ Are all I/O exceptions handled in a reasonable way?

9. Module Interface Defects (MI)

☑ Are the number, order, types, and values of parameters in every method call in

agreement with the called method ́s declaration?

☑ Do the values in units agree (e.g., inches versus yards)?

❑ If an object or array is passed, does it get changed, and changed correctly

by the called method?

10. Comment Defects (CD)

☑ Does every method, class, and file have an appropriate header comment?

❑ Does every attribute, variable, and constant declaration have a comment?

☑ Is the underlying behavior of each method and class expressed in plain language?

☑ Is the header comment for each method and class consistent with the behavior of

the method or class?

☑ Do the comments and code agree?

☑ Do the comments help in understanding the code?

☑ Are there enough comments in the code?

❑ Are there too many comments in the code?

11. Layout and Packaging Defects (LP)

☑ Is a standard indentation and layout format used consistently?

☑ For each method: Is it no more than about 60 lines long?

❑ For each compile module: Is no more than about 600 lines long?

12. Modularity Defects (MD)

☑ Is there a low level of coupling between modules (methods and classes)?

☑ Is there a high level of cohesion within each module (methods or class)?

❑ Is there repetitive code that could be replaced by a call to a method that provides

the behavior of the repetitive code?

☑ Are the Java class libraries used where and when appropriate?

13. Storage Usage Defects (SU)

☑ Are arrays large enough?

❑ Are object and array references set to null once the object or array is no longer

needed?

14. Performance Defects (PE)

☑ Can better data structures or more efficient algorithms be used?

❑ Are logical tests arranged such that the often successful and inexpensive tests

precede the more expensive and less frequently successful tests?

☑ Can the cost of re-computing a value be reduced by computing it once and storing

the results?

☑ Is every result that is computed and stored actually used?

❑ Can a computation be moved outside a loop?

❑ Are there tests within a loop that do not need to be done?

❑ Can a short loop be unrolled?

❑ Are there two loops operating on the same data that can be combined into one?

❑ Are frequently used variables declared register?

☑ Are short and commonly called methods declared inline?

# Found defects

Não foram encontrados qualquer tipo de defeitos que possam comprometer o funcionamento correcto da aplicação pois esta é a versão final para entregar ao “Product Owner”.

# Corrective measures

*Nao existem.*

# Conclusions of the inspection process

*O trabalho realizado pelo nosso grupo (31) pela disciplina de Engenharia de Software I foi implementado correctamente sem erros como podemos demonstrar. O resultado demonstra exatamente o que era pedido pelo “Product Owner”. Todos os ficheiros provenientes do algoritmo podem ser analisados tal como Tabela, Gráfico etc.*