|  |  |
| --- | --- |
|  | *Code Inspection Report*  *Anti-Spam Configuration Software Development Project*  BSc in [LEI | LIGE]  Academic Year 2017/2018 - 1º Semester  Software Engineering I  Group Id ...  66049, Adolfo Moreno, LEI-PL  69540, Joana Mata, LIGE-PL  66144, João Mendonça, LEI-PL  68747, Raquel Martins, LIGE-PL  ISCTE-IUL, Instituto Universitário de Lisboa  1649-026 Lisbon  Portugal  November 25th 2017 |

**Table of Contents**

[Introduction 3](#_Toc501677922)

[Code inspection – Gui 3](#_Toc501677923)

[Code inspection – ConfigurationWindow 3](#_Toc501677924)

[Code inspection – EditWindow 4](#_Toc501677925)

[Code inspection checklist 4](#_Toc501677926)

[Found defects 7](#_Toc501677927)

[Corrective measures 7](#_Toc501677928)

[Conclusions of the inspection process 7](#_Toc501677929)

# Introduction

A bad anti-spam filtering service allows user email boxes to be flooded of spam messages, turns email service useless or of extremely low productivity, to that end Anti-SPAM filtering must allow frequent addition of rules to match user preferences (language, area of activity, professional/leisure mailbox, etc.) and adapt to new/volatile contents. The software we developed will allow a user to, after choosing the paths to rules.cf(With real-valued rules ranging [-5,5] and to a set of reference spam and ham email messages (files ham.log and spam.log). Then through a framework we developer, a user can either manually configure the weights of the spam filtering rules or generate it automatically. Then, we compare the obtained rules to the spam and ham log files and the program tells you the amount of False Positives and False Negatives. The objective of the program is being able to calculate the optimal configuration that allows for the minimum amount of False Positives and False Negatives. The application also brings the feature to create Random Configurations or to Manually change the Configuration, and the Optimal Configuration will account for the end-user’s mailbox usage: Professional, Hobby or Mixed.

# Code inspection – Gui

The main user interface that allows the user to define the paths to rules.cf, ham.log and spam.log. Once the paths are correctly assigned, it moves on to the ConfigurationWindow

|  |  |
| --- | --- |
| *Meeting date:*  *Meeting duration:*  *Moderator:*  *Producer:*  *Inspector:*  *Recorder:* | *22/12/2017*  *10 minutes*  *João Mendonça*  *Raquel Martins*  *Adolfo Moreno*  *Joana Mata* |
| *Component name (Package/Class/Method):* | *Gui.java* |
| *Component was compiled:* | Yes |
| *Component was executed:* | Yes |
| *Component was tested without errors:* | Yes |
| *Testing coverage achieved:* | 93,4% |

# Code inspection – ConfigurationWindow

The configuration window allows you to Generate a random configuration, edit the rule weights directly, to calculate the amount of FPs and FNs or to generate an optimal configuration.

|  |  |
| --- | --- |
| *Meeting date:*  *Meeting duration:*  *Moderator:*  *Producer:*  *Inspector:*  *Recorder:* | *22/12/2017*  *10 minutes*  *João Mendonça*  *Raquel Martins*  *Adolfo Moreno*  *Joana Mata* |
| *Component name (Package/Class/Method):* | ConfigurationWindow.java |
| *Component was compiled:* | Yes |
| *Component was executed:* | Yes |
| *Component was tested without errors:* | Yes |
| *Testing coverage achieved:* | 61,9% |

# Code inspection – EditWindow

A button that allows the user to configure the rule weights manually.

|  |  |
| --- | --- |
| *Meeting date:*  *Meeting duration:*  *Moderator:*  *Producer:*  *Inspector:*  *Recorder:* | *22/12/2017*  *10 minutes*  *João Mendonça*  *Raquel Martins*  *Adolfo Moreno*  *Joana Mata* |
| *Component name (Package/Class/Method):* | EditWindow.java |
| *Component was compiled:* | Yes |
| *Component was executed:* | Yes |
| *Component was tested without errors:* | Yes |
| *Testing coverage achieved:* | 70,4% |

**Code inspection** **– FileChooser**

|  |  |
| --- | --- |
| *Meeting date:*  *Meeting duration:*  *Moderator:*  *Producer:*  *Inspector:*  *Recorder:* | *22/12/2017*  *10 minutes*  *João Mendonça*  *Raquel Martins*  *Adolfo Moreno*  *Joana Mata* |
| *Component name (Package/Class/Method):* | FileChooser.java |
| *Component was compiled:* | Yes |
| *Component was executed:* | Yes |
| *Component was tested without errors:* | Yes |
| *Testing coverage achieved:* | 89,8% |

# Code inspection checklist

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

☑ Are 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 variables 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 local 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 (CD)

☑ Does each 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. Computation/Numeric Defects (CN)

Are there any computations with mixed data types?

Is overflow or underflow possible during a computation?

☑ For each expressions 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 (CM)

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 (MO)

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 recomputing 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 short and commonly called methods declared inline?

# Found defects.

|  |  |  |  |
| --- | --- | --- | --- |
| **Found defect Id** | **Package, Class, Method, Line** | **Defect category** | **Description** |
| 1 | EditWindow.java | CF | Handling Exceptions could be improved |
| 2 | Package Gui | CM | More documentation and Comments could be provided |

# Corrective measures

*ParseDouble from user input can give a NumberFormatException and should be handled.*

*Correct CM defects through detailed documentation over each method.*

**Code inspection – AntiSpamFilesReader**

Reads the files that the Anti-Spam filter generates.

|  |  |
| --- | --- |
| *Meeting date:*  *Meeting duration:*  *Moderator:*  *Producer:*  *Inspector:*  *Recorder:* | *22/12/2017*  *10 minutes*  *João Mendonça*  *Adolfo Moreno*  *Raquel Martins*  *Joana Mata* |
| *Component name (Package/Class/Method):* | AntiSpamFilesReader.java |
| *Component was compiled:* | Yes |
| *Component was executed:* | Yes |
| *Component was tested without errors:* | Yes |
| *Testing coverage achieved:* | 94,4% |

**Code inspection – LogReader**

Reads .log files.

|  |  |
| --- | --- |
| *Meeting date:*  *Meeting duration:*  *Moderator:*  *Producer:*  *Inspector:*  *Recorder:* | *22/12/2017*  *10 minutes*  *João Mendonça*  *Adolfo Moreno*  *Raquel Martins*  *Joana Mata* |
| *Component name (Package/Class/Method):* | LogReader.java |
| *Component was compiled:* | Yes |
| *Component was executed:* | Yes |
| *Component was tested without errors:* | Yes |
| *Testing coverage achieved:* | 95,1% |

**Code inspection – ReadConfiguration**

Reads rules.cf and creates an HashMap<Rule,Value>.

|  |  |
| --- | --- |
| *Meeting date:*  *Meeting duration:*  *Moderator:*  *Producer:*  *Inspector:*  *Recorder:* | *22/12/2017*  *10 minutes*  *João Mendonça*  *Adolfo Moreno*  *Raquel Martins*  *Joana Mata* |
| *Component name (Package/Class/Method):* | ReadConfiguration.java |
| *Component was compiled:* | Yes |
| *Component was executed:* | Yes |
| *Component was tested without errors:* | Yes |
| *Testing coverage achieved:* | 89,9% |

# Code inspection checklist

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

Are 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 variables 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 local 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 (CD)

☑ Does each 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. Computation/Numeric Defects (CN)

Are there any computations with mixed data types?

Is overflow or underflow possible during a computation?

☑ For each expressions 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 (CM)

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 (MO)

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 recomputing 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 short and commonly called methods declared inline?

# Found defects.

|  |  |  |  |
| --- | --- | --- | --- |
| **Found defect Id** | **Package, Class, Method, Line** | **Defect category** | **Description** |
| 1 | ReadConfiguration.java | CF | Handling Exceptions could be improved |
| 2 | Package Reader | CM | More documentation and Comments could be provided |

# Corrective measures

*Could handle what would happen if path was null on the method readFile.*

*Correct CM defects through detailed documentation over each method.*

**Code inspection – DetectionCalculator**

Counts the number of False Positives and False Negatives with a set configuration.

|  |  |
| --- | --- |
| *Meeting date:*  *Meeting duration:*  *Moderator:*  *Producer:*  *Inspector:*  *Recorder:* | *22/12/2017*  *10 minutes*  *João Mendonça*  *Adolfo Moreno*  *Raquel Martins*  *Joana Mata* |
| *Component name (Package/Class/Method):* | DetectionCalculator.java |
| *Component was compiled:* | Yes |
| *Component was executed:* | Yes |
| *Component was tested without errors:* | Yes |
| *Testing coverage achieved:* | 100% |

**Code inspection – OptimalCalculator**

Using the NSGA2 algorithm provided it tries to calculate the optimal configuration, meaning, the configuration of rule weights that result in the least False Negatives and False Positives combined.

|  |  |
| --- | --- |
| *Meeting date:*  *Meeting duration:*  *Moderator:*  *Producer:*  *Inspector:*  *Recorder:* | *22/12/2017*  *10 minutes*  *João Mendonça*  *Adolfo Moreno*  *Raquel Martins*  *Joana Mata* |
| *Component name (Package/Class/Method):* | OptimalCalculator.java |
| *Component was compiled:* | Yes |
| *Component was executed:* | Yes |
| *Component was tested without errors:* | Yes |
| *Testing coverage achieved:* | 100% |

# Code inspection checklist

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

Are 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 variables 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 local 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 (CD)

☑ Does each 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. Computation/Numeric Defects (CN)

Are there any computations with mixed data types?

Is overflow or underflow possible during a computation?

☑ For each expressions 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 (CM)

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 (MO)

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 recomputing 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 short and commonly called methods declared inline?

# Found defects.

|  |  |  |  |
| --- | --- | --- | --- |
| **Found defect Id** | **Package, Class, Method, Line** | **Defect category** | **Description** |
| 1 | DetectionCalculator.java | CF | Handling Exceptions could be improved |
| 2 | Package Utils | CM | More documentation and Comments could be provided |

# Corrective measures

*When doing Double.parseDouble() it could be handled when the string parsed is in incorrect form.*

*Correct CM defects through detailed documentation over each method.*

**Code inspection – AntiSpamFilterProblem**

It’s the proposed spam-filter problem that the software tried to solve.

|  |  |
| --- | --- |
| *Meeting date:*  *Meeting duration:*  *Moderator:*  *Producer:*  *Inspector:*  *Recorder:* | *22/12/2017*  *10 minutes*  *João Mendonça*  *Raquel Martins*  *Adolfo Moreno*  *Joana Mata* |
| *Component name (Package/Class/Method):* | AntiSpamFilterProblem.java |
| *Component was compiled:* | Yes |
| *Component was executed:* | Yes |
| *Component was tested without errors:* | Yes |
| *Testing coverage achieved:* | 100% |

**Code inspection – AntiSpamFilterAutomaticConfiguration**

Runs an automatic configuration that tries to calculate the best way to solve the FP and FN problem.

|  |  |
| --- | --- |
| *Meeting date:*  *Meeting duration:*  *Moderator:*  *Producer:*  *Inspector:*  *Recorder:* | *22/12/2017*  *10 minutes*  *João Mendonça*  *Adolfo Moreno*  *Raquel Martins*  *Joana Mata* |
| *Component name (Package/Class/Method):* | AntiSpamFilterAutomaticConfiguration.java |
| *Component was compiled:* | Yes |
| *Component was executed:* | Yes |
| *Component was tested without errors:* | Yes |
| *Testing coverage achieved:* | 98,3% |

# Code inspection checklist

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

Are 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 variables 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 local 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 (CD)

☑ Does each 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. Computation/Numeric Defects (CN)

Are there any computations with mixed data types?

Is overflow or underflow possible during a computation?

☑ For each expressions 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 (CM)

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 (MO)

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 recomputing 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 short and commonly called methods declared inline?

# Found defects.

|  |  |  |  |
| --- | --- | --- | --- |
| **Found defect Id** | **Package, Class, Method, Line** | **Defect category** | **Description** |
| 1 | AntiSpamFilterProblem.java | PE | Double array data structure could be improved |
| 2 | Package antiSpamFilter | CM | More documentation and Comments could be provided |

# Corrective measures

*The double[] array could be an ArrayList<Double>, making the data structure more efficient and fail proof, and will be implemented by João Mendonça on 22/12/2017.*

*Correct CM defects through detailed documentation over each method.*

# Conclusions of the inspection process

Overall for what’s intended, the code doesn’t have any critical flaw and runs well, without errors. Some exceptions could be better handled and the methods better documented, but it’s not critical for the software, no real changes are needed besides the one that will be corrected.