Code Inspection Report

Anti-Spam Configuration Software Development Project

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

Grupo 67
73372, Guilherme Pereira, IC1
72806, Ricardo Silva, IC2
73313, Pedro Azevedo, IC1
64545, André Santos, I-PLC1

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

December 22th 2017

Table of Contents

Introduction 3
Code Inspection - Package antiSpamFilter 3
Code Inspection Checklist 4
Found Defects 6
Corrective Measures 6
Conclusions of the inspection process 7

Introduction

A distribuição de informação maliciosa e ilícita do serviço de email é normalmente designada como spam. Os utilizadores deste serviço estão constantemente a receber mensagens não solicitadas que prejudicam a sua produtividade pessoal e põe em risco a segurança dos sistemas, serviços e informação dos utilizadores do serviço. Estudos revelam que 95% a 97% das mensagens de email são spam, sendo que um bom serviço de filtragem anti-spam não pode permitir que uma caixa de correio seja inundada com mensagens de spam, pois acabaria por tornar o serviço de e-mail inútil ou de baixa produtividade.

Sendo assim, um sistema de filtro anti-spam deve permitir o acréscimo frequente de regras de acordo com as preferências do utilizador (linguagem, área de atividade, caixa de correio profissional/lazer, etc.). Estas regras devem ser calibradas, ou seja, devem ser-lhes atribuídas pesos, por forma a se ter conhecimento da quantidade de mensagens que deve ser classificada como legítima (designada de ham) ou ilegítima (designada de spam).

Deste modo, este projeto, no âmbito da unidade curricular de Software Engeneering I do 3º ano da Licenciatura em Informática e Gestão de Empresas, consiste no desenvolvimento de um software que proporcione uma configuração ótima do serviço de filtragem que minimize a ocorrência de classificações de mensagens como spam quando estas são legítimas (FP - Falsos Positivos), uma vez que o filtro anti-spam, no nosso caso, é destinado a caixas de email para uso profissional.

Este documento consiste num relatório da inspeção de código realizada ao software implementado em java.

Code Inspection - Package antiSpamFilter

Meeting date: Meeting duration: Moderator: Producer: Inspector: Recorder:	21/12/2017 30 minutes Guilherme Pereira Guilherme Pereira Ricardo Silva Ricardo Silva
Component name (Package/Class/Method):	Package antiSpamFilter
Component was compiled:	Sim
Component was executed:	Sim
Component was tested without errors:	Sim
Testing coverage achieved:	Sim

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?
- na Do any subclasses have common members that should be in the superclass?
- na 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?
- na Has an "&" inadvertently been interchanged with a "&&" or a "|" for a "||"?

7. Control Flow Defects (CF)

- [x] For each loop: Is the best choice of looping constructs used?
- [x] Will all loops terminate?
- [ ] na When there are multiple exits from a loop, is each exit necessary and handled properly?
- [ ] na Does each switch statement have a default case?
- [ ] na Are missing switch case break statements correct and marked with a comment?
- [ ] na Do named break statements send control to the right place?
- [x]
- [ ] Is the nesting of loops and branches too deep, and is it correct?
- [ ] na Can any nested if statements be converted into a switch statement?
- [ ] na Are null bodied control structures correct and marked with braces or comments?

- [x] Are all exceptions handled appropriately?
- [x] Does every method terminate?

8. Input-Output Defects (IO)

- [x] Have all files been opened before use?
- [x] Are the attributes of the input object consistent with the use of the file?
- [x] Have all files been closed after use?
- [x] Are there spelling or grammatical errors in any text printed or displayed?
- [x] Are all I/O exceptions handled in a reasonable way?

9. Module Interface Defects (MI)

- [x] Are the number, order, types, and values of parameters in every method call in agreement with the called method's declaration?
- [x] Do the values in units agree (e.g., inches versus yards)?
- [x] If an object or array is passed, does it get changed, and changed correctly by the called method?

10. Comment Defects (CM)

- [x] Does every method, class, and file have an appropriate header comment?
- [x] Does every attribute, variable, and constant declaration have a comment?
- [x] Is the underlying behavior of each method and class expressed in plain language?
- [x] Is the header comment for each method and class consistent with the behavior of the method or class?
- [x] Do the comments and code agree?
- [x] Do the comments help in understanding the code?
- [x] Are there enough comments in the code?
- [x] Are there too many comments in the code?

11. Layout and Packaging Defects (LP)

- [x] Is a standard indentation and layout format used consistently?
- [x] For each method: Is it no more than about 60 lines long?
- [x] For each compile module: Is no more than about 600 lines long?

12. Modularity Defects (MO)

- [x] Is there a low level of coupling between modules (methods and classes)?
- [x] Is there a high level of cohesion within each module (methods or class)?
- [x] Is there repetitive code that could be replaced by a call to a method that provides the behavior of the repetitive code?
- [x] 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 longerneeded?

14. Performance Defects (PE)

☑ Can better data structures or more efficient algorithms be used?
na 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 frequently used variables declared register?
☑ Are short and commonly called methods declared inline?

Nota: na significa 'não se aplica

Found Defects

Found Defect ID	Package, Class, Method, Line	Defect category	Description
1	Package antiSpamFilter, Automatic Configuration, constructsAndOperateOnButtonCalculateNewFPeFN, 203	Method Definition Defects (FD)	O valor do parâmetro 'table' passado no método em questão não é verificado antes de ser utilizado.
2	Package antiSpamFilter	Comment Defects (CM)	Os métodos possuem comentários Javadoc, no entanto as classes não.
3	Package antiSpamFilter	Comment Defects (CM)	As variáveis, atributos e constantes não possuem comentários javadoc.
4	Package antiSpamFilter, AntiSpamFilterProblem, evaluate	Modularity Defects (MO)	O cálculo do número de Falsos Positivos e Falsos Negativos talvez pudesse ser realizado invocando as funções já implementadas na classe Manual Configuration
5	Package antiSpamFilter	Storage Usage Defects (SU)	As referências de objetos e arrays não são colocadas a null quando deixam de ser necessárias.

Corrective Measures

A reunião de Code Inspection foi realizada apenas um dia antes da entrega do projeto, na qual apenas compareceram dois elementos do grupo. Foi, portanto, uma má decisão, uma vez que deveríamos ter realizado uma reunião de Code Inspection no final do Sprint1, por exemplo, por forma a analisar de uma forma detalhada o código desenvolvido e ter a perceção dos erros cometidos, para poder melhorar os aspetos menos positivos no futuro.
Desta forma, os defects que foram detetados nesta reunião não puderam ser corrigidos a tempo da entrega do projeto.

Conclusions of the inspection process

Após uma análise detalhada do software desenvolvido, podemos verificar que, em regra, foram cumpridas as normas da programação, sendo que se identificaram poucos defeitos no mesmo. No entanto, tal como referido no ponto anterior, deveria ter sido realizada uma reunião de Code Inspection anteriormente, por forma a não existirem estes erros no projeto. No entanto, é possível concluir que o projeto, apesar de não estar 100% otimizado, cumpre de forma eficiente os requisitos, necessitando apenas de serem implementadas algumas medidas corretivas para eliminar os defeitos identificados e aumentar a qualidade do software.