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 ID **28**
54431, Bruno Gama, EIC1
68640, André Sousa, EIC1
68674, Rafael Fernandes, EIC1
64592, Rui Farinha, IC2

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

Table of Contents

Introduction

Nos dias de hoje, a internet desempenha um papel importantíssimo no nosso dia-a-dia. Segundo um estudo realizado pelo Radicati Group, cerca de 3,7 mil milhões de pessoas usam o email, seja para fins profissionais ou de lazer. Por dia, cerca de 95% dos emails que circulam na rede são **SPAM** e, por isso, é necessário que as companhias que prestam o serviço de email tenham um filtro de spam adequado.

Na unidade curricular de **Engenharia de Software I** foi nos proposto a gestão de um projeto de produção de software. O software pedido consiste num filtro de Anti-Spam para emails profissionais e que gera um tratamento para os emails de spam, que podem ser regulados manualmente ou automaticamente. Neste relatório demonstramos o resultado da reunião de inspeção do código do software produzido.

Code inspection – Name of the component being inspected

A reunião realizou-se no dia 19 de Dezembro de 2017, durou aproximadamente 2 horas e os elementos do grupo desempenharam as seguintes funções:

Função	**Elementos do grupo**
Moderator	Bruno Gama
Producer	André Sousa, Rafael Fernandes
Inspector	Bruno Gama, André Sousa, Rafael Fernandes, Rui Farinha
Recorder	Rui Farinha

A realização do code inspection foi elaborada nos seguintes componentes do software:

Nome do Componente	**Compilado?**	**Executado?**	**Testado sem erros**	**Cobertura de testes obtida**
<antiSpamFilter> AntiSpamFilter.java	Sim	Sim	0 erros	89.7%
<antiSpamFilter.gui> Gui.java	Sim	Sim	0 erros	69.6%
<antiSpamFilter.utils> Rule.java	Sim	Sim	0 erros	100%

Code inspection checklist

A checklist usada para este projeto foi-nos fornecida como material para a realização do projeto da unidade curricular, Engenharia de Software I.

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 frequently used variables declared register?	✓
Are short and commonly called methods declared inline?	✓

Found defects

Found defect ID	Package, Class, Method, Line	Defect category	Description
1	<antiSpamFilter.gui> Gui.java	Comment Defects	Alguns métodos da classe Gui não têm os comentários necessários.
2	<antiSpamFilter.gui> Gui.java	Layout and Packaging Defects	Existem métodos com um número de linhas de código superior a 60.
3	<antiSpamFilter> AntiSpamFilter.java	Variable, Attribute, and Constant Declaration Defects	Existem algumas variáveis que tem nomes similares.

Corrective measures

Com a inspeção ao código do software desenvolvido que realizamos e, tendo em conta, que nos testes não nos surgiu nenhum erro apenas encontramos 3 defeitos no nosso código.

Foi deliberado que no terceiro sprint deste projeto, um dos objetivos seria o tratamento destes defeitos no código, efetuando uma limpeza no código.

Found Defect ID	Corrections
1	Clarificar os métodos da classe Gui, responsável pelo interface, através de alguns comentários necessários.
2	Tentar reduzir as linhas de códigos repartindo em vários métodos e juntado-os num método só.
3	Renomear essas variáveis com nomes mais concretos e/ou diferentes de outras variáveis.

Conclusions of the inspection process

Com este processo, frequentemente, realizado em Engenharia de Software fomos capazes de melhorar o nosso código, na sua compreensão através de mais e melhores comentários e, ainda, de algumas melhorias na nomenclatura das variáveis. Para além disso, tivemos a oportunidade de fazer uma total análise ao código que produzimos para o projeto. Após esta análise procedemos a algumas mudanças no código mas que devido à boa programação já estava feita, foram simples e não necessitámos de muito tempo para as executar.

Terminada esta tarefa, podemos concluir que o nosso projeto vai ao encontro dos requisitos pedidos e, também, cumpre com grande parte das “regras” da Engenharia de Software.