Code Inspection Report

'Bom Dia Academia' Software Development Project

BSc/MSc in [LEI | LIGE | METI]
Academic Year 2018/2019 - 1º Semester
Software Engineering I

Grupo 114
69969, João Oliveira, METI A1
69987, José Santos, METIA1
73062, Ricardo Castro, METI A1
73431, Diogo Sousa, METI A1

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

November 2018

Table of Contents

Introduction

Bom Dia Academia é um software desenvolvido pelo grupo 114 que, através de uma interface gráfica, reúne as redes sociais Facebook e Twitter bem como um serviço de e-mail. Esta interface permite uma interação facilitada das redes tanto na visualização de conteúdo como na partilha ou envio de novas mensagens ou publicações.

Code inspection – Package do Twitter

Este package permite que se faça a interação com o Twitter, nomeadamente ler e tratar a Timeline bem como postar tweets.

Meeting date:	*3/12/2018*
Meeting duration:	*60 minutes*
Moderator:	*José Santos*
Producer:	*João Oliveira*
Inspector:	*José Santos*
Recorder:	*José Santos*
Component name (Package/Class/Method):	*Twitter*
Component was compiled:	*Yes*
Component was executed:	*Yes*
Component was tested without errors:	*Yes*
Testing coverage achieved:	*88%*

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

Found defects

Identify and describe found defects, opinions and suggestions.

Found defect Id	Package, Class, Method, Line	Defect category	Description
1	Twitter, TwitterApp, fetchTimeLine(), 99	CF	A exceção não era tratada corretamente.
2	Twitter, TwitterApp, 25, 26	VC	Alguns atributos eram inicializados fora dos métodos quando na realidade só eram utilizados em um método.

Corrective measures

Os erros detetados foram corrigidos no momento.

Conclusions of the inspection process

Após a realização da inspeção foi possível concluir que o pacote associado ao Twitter tratava informação pouco complexa, sendo que os únicos aspetos a melhorar encontrados estão descritos na tabela acima. Como se tratavam de falhas simples, foram corrigidas na hora pelo autor da classe.

Code inspection – Package da Interface

Este package permite a interação gráfica entre todas as interfaces desenvolvidas.

Meeting date:	*7/12/2018*
Meeting duration:	*105 minutes*
Moderator:	*José Santos*
Producer:	*João Oliveira*
Inspector:	*José Santos*
Recorder:	*José Santos*
Component name (Package/Class/Method):	*Interface*
Component was compiled:	*Yes*
Component was executed:	*Yes*
Component was tested without errors:	*Yes*
Testing coverage achieved:	-

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 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	Interface, FaceEditor, line 33	VC	Uma variável não local podia ser local
2	Interface, interface, sendNot(), line 245	FD	O nome deste método não está de acordo com a sua funcionalidade
3	Interface, interface, iconsAction(), line 300	MO	Este método apresenta código repetitivo

Corrective measures

A variável não local foi corrigida para variável local;

O método que apresentava um nome que não estava de acordo com a respetiva funcionalidade foi corrigido, passando a denominar-se “openEditor()”;

O último defeito encontrado ainda não foi corrigido. Tratava-se de uma função que associa cada interface às listas respetivas de timeline. A maneira de corrigir este erro seria, por exemplo, fazer uma função que recebesse como parâmetros a interface e a lista e os associasse, para posteriormente invocar.

Quem tratará deste erro será o José Santos.

Conclusions of the inspection process

Após a realização da inspeção foi possível concluir que o pacote associado à Interface possui algumas classes e é um pouco extenso. No entanto, não apresenta muitos erros nem graves e funciona conforme pretendido.

Code inspection – Package do Facebook

Este package permite a captação da timeline e outras interações com o Facebook

Meeting date:	*5/12/2018*
Meeting duration:	*60 minutes*
Moderator:	*José Oliveira*
Producer:	*Ricardo Castro*
Inspector:	*Diogo Sousa*
Recorder:	*João Oliveira*
Component name (Package/Class/Method):	*Facebook*
Component was compiled:	*Yes*
Component was executed:	*Yes*
Component was tested without errors:	*Yes*
Testing coverage achieved:	*85,7%*

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 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	Facebook, PostGroups, PostGroups, 34	FD	Método descontinuado mas funcional
2	Facebook, PostGroups, PostarImagem,68	FD	Método descontinuado mas funcional
3	Facebook, Extend, Extend, 22	FD	Método descontinuado mas funcional
4	Facebook, Timeline, Timeline, 27	FD	Método descontinuado mas funcional

Corrective measures

Para todos os ids, deveriam ser alteradas as funções para outras com funcionalidade semelhante mas que fossem suportadas atualmente. Isto deveria ser corrigido no próximo sprint mas como este não irá existir o defeito não será corrigido.

Conclusions of the inspection process

O código do facebook está funcional e corresponde às expetativas postas pelo grupo para o seu fincionamento. Os métodos descontinuados podiam ser trocados para outros com suporte mas funcionam perfeitamente e não são motivo de preocupação. No geral o código está bem estruturado.

Code inspection – Package do Email

Este package permite a interação com o e-mail.

Meeting date:	*3/12/2018*
Meeting duration:	*90 minutes*
Moderator:	*José Santos*
Producer:	*Diogo Sousa*
Inspector:	*Ricardo Castro*
Recorder:	*José Santos*
Component name (Package/Class/Method):	*Mail*
Component was compiled:	*Yes*
Component was executed:	*Yes*
Component was tested without errors:	*Yes*
Testing coverage achieved:	*83,5%*

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 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	Mail, MailHandler, receberEmail	VC	Alguns e-mails vêm desformatados

Corrective measures

Não será possível corrigir o erro, visto que não conseguimos identificar o que provoca a desformatação dos e-mails.

Conclusions of the inspection process

De um modo geral, o código está bem estruturado e bem otimizado. Todas as classes deste Package têm um header a explicar a sua função, bem como todos os métodos têm explicado detalhadamente os atributos que recebem, o seu output e a sua função.

Code inspection – Package do Common

Este package tem duas classes: StandardInfoStruct e XML. A primeira classe permite obter uma estrutura geral para os tipos de mensagens utilizados enquanto que a segunda permite a interação com o ficheiro XML, nomeadamente ler e escrever.

Meeting date:	*3/12/2018*
Meeting duration:	*60 minutes*
Moderator:	*João Oliveira*
Producer:	*José Santos, Diogo Sousa*
Inspector:	*João Oliveira*
Recorder:	*Ricardo Castro*
Component name (Package/Class/Method):	*Common*
Component was compiled:	*Yes*
Component was executed:	*Yes*
Component was tested without errors:	*Yes*
Testing coverage achieved:	*84,9%*

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

Found defectss

Found defect Id	Package, Class, Method, Line	Defect category	Description
1	Common, Xml, leituraXML, line 120	CF	Uma exceção não é tratada.

Corrective measures

Foi aplicada a exceção adequada ao caso.

Conclusions of the inspection process

Este package não foi exigente na inspeção do código, no entanto o erro encontrado permitiu um melhor desempenho na aplicação para o caso em que a exceção é utilizada.