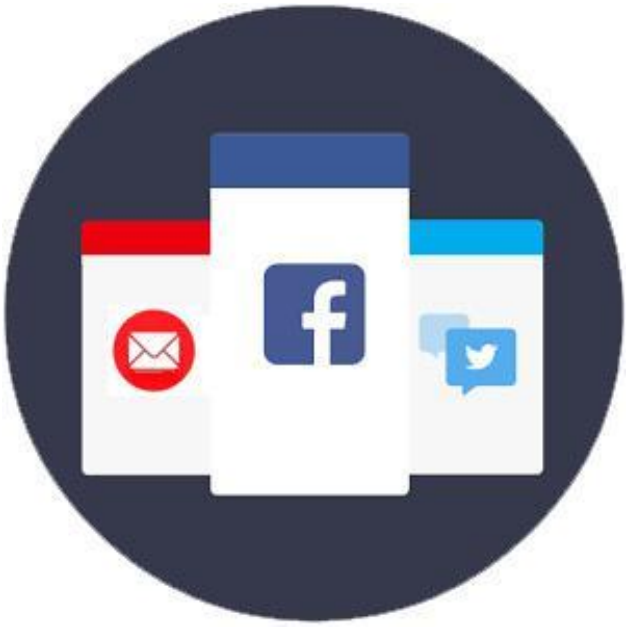
Code Inspection report

'Bom dia Academia' Software Development Project



Group 11:
77586 Rúben Silva, EIC2
77624 Matias Correia, EIC2
77755 Leonardo Rosa, EIC2
77761 Rodrigo Tavares, EIC2

BSc/MSc in [LEI|LIGE|METI]
Academic Year 2018/2019 – 1st Semester
Software Engineering 1

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

November 2018

Table of Contents

Conteúdo

Introduction

The main goal of this software is to provide the user a pleasant and simple experience while offering the change to check his social media feeds in one single area.

'Bom dia Academia' gathers the feed of your email, facebook and twitter accounts in the same app allowing you to use all the main features of each one.

As conveyed above one of our main objectives throughout the development process of this software was to give the user a pleasant and simple experience. For that purpose we focused heavily on making the GUI appealing and making every single button intuitive and user friendly.

This software implements twitter 4j, restfb and javax libraries. It consists in retrieving and placing information (tweets, posts, mails) from the email and respective social networks. Although it was only supposed to get academic information we turned up a notch and implemented features that we found interesting and useful. Some of the features we implemented (besides filters, offline use and many more) are, for example, instead of retrieving all of the possible tweets/posts/mails we implemented an ability to retrieve more posts with a button "more" (Paging mechanism).
[PRINT DA APP COM UMA APP E O BUTAO DO MORE SEM DARK THEME]

Efficiency is one of the most important things in any application. With that being said, we use threads to buffer the list that the "more" button provides. Additionally, we also use threads to retrieve the lists that the GUI needs (and not the GUI thread) so it won't freeze.

Another important aspect is that our software has its own user database which means that we are able to store the settings defined by each user and load them when they log on again.

In addition, we tried as hard as we could to get a different, more modern and most important, user-friendly GUI that bounds it all together.

[PRINT DA APP COM 2 APPS COM DARK THEME]
As it demonstrates on the diagram (generated by modelgoon) and as we said previously, the GUI binds it all together.

Code inspection – Facebook

This class interacts with the API restFb and retrieves the posts associated to a token

[AQUI EM TODAS (DEBAIXO DA DESCRICAO COLOCAR O DIAGRAMA DE CLASSES]

Meeting date: *Meeting duration:* *Moderator:* *Producer:* *Inspector:* *Recorder:*	*7/12/2017* *90 minutes* *Rodrigo Tavares* *Leonardo Rosa* *Matias Correia* *Rúben Silva*
Component name (Package/Class/Method):	*FacebookApp*
Component was compiled:	*Yes*

Component was executed:	*Yes*
Component was tested without errors:	*Yes*
Testing coverage achieved:	91.4

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?
(not implemented) Do any subclasses have common members that should be in the superclass?
(not implemented) 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 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?
(no switchs used) Does each switch statement have a default case?
(no switchs used) Are missing switch case break statements correct and marked with a comment?
(no breaks set) 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) (not used any 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	apps, FacebookApp.java,, line 34		Class local variable wrongly set to static
2	apps,FacebookApp.java, getTimeLine() and getTimeLine(String s), lines 57 and 87 respectively		Redundant Code used in two methods

Corrective measures

Change the local variable to non-static and instead of having two methods have just one checking the parameter if it is a String or a null.

Conclusions of the inspection process

Overall the code is in good shape meting the standards, working as intendend, with no detected fails so is set to be delivered to the client.

Code inspection – Twitter

This class interacts with the API twitter4j and interacts with the account related to the given tokens

Meeting date: *Meeting duration:* *Moderator:* *Producer:* *Inspector:* *Recorder:*	*7/12/2017* *60 minutes* *Matias Correia* *Rúben Silva* *Leonardo Rosa* *Rodrigo Tavares*
Component name (Package/Class/Method):	*TwiterApp*
Component was compiled:	*Yes*
Component was executed:	*Yes*
Component was tested without errors:	*Yes*
Testing coverage achieved:	88,8

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?
(not implemented) Do any subclasses have common members that should be in the superclass?
(not implemented) 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 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?
(no switchs used) Does each switch statement have a default case?
(no switchs used) Are missing switch case break statements correct and marked with a comment?
(no breaks set) 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) (not used any 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	apps, TwitterApp.java, getTimeLine() and getTimeLine(String s), lines 57 and 87 respectively		Redundant Code used in two methods

Corrective measures

Instead of having two methods have just one checking the parameter if it is a String or a null.

Conclusions of the inspection process

This class displays a very simple code that connects to the twitter API and gets access to twitter features. It is ready to be sent to the client.

Code inspection – Email

This class retrieves the emails and lets you replay and send emails.

Meeting date: *Meeting duration:* *Moderator:* *Producer:* *Inspector:* *Recorder:*	*7/12/2017* *60 minutes* *Leonardo Rosa* *Matias Correia* *Rodrigo Tavares* *Rúben Silva*
Component name (Package/Class/Method):	*EmailApp*
Component was compiled:	*Yes*
Component was executed:	*Yes*
Component was tested without errors:	*Yes*
Testing coverage achieved:	75.4

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?
(not implemented) Do any subclasses have common members that should be in the superclass?
(not implemented) 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 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?
(no switchs used) Does each switch statement have a default case?
(no switchs used) Are missing switch case break statements correct and marked with a comment?
(no breaks set) 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) (not used any 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

Corrective measures

This code should have more useful comments implements and more organized loops and conditions.

Conclusions of the inspection process

Despite the code lacking useful comments and some conditions looking like they could be a little bit more efficient and maybe less redundant the code works well and fulfills entirely his functions.

Code inspection – apps (package)

It's not a typo, the package name is always in lower case. This package has all of the tools related with the GUI.
[ESTE É RESPETIVO AO PACOTE , POR ISSO PACKAGE DIAGRAM]

Meeting date: *Meeting duration:* *Moderator:* *Producer:* *Inspector:* *Recorder:*	*7/12/2017* *92 minutes* *Leonardo Rosa* *Rodrigo Tavares* *Rúben Silva* *Matias Correia*
Component name (Package/Class/Method):	*gui*
Component was compiled:	*Yes*
Component was executed:	*Yes*
Component was tested without errors:	*Yes*
Testing coverage achieved:	*JUnit was not implemented in any of the classes that belong to this package because there are a lot of branches (catches, ifs, …)*

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?
(Not implemented)Does each switch statement have a default case?
(Not implemented)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) (not used any 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?
(Not implemented).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? NO
×Are there tests within a loop that do not need to be done? NO
✓Can a short loop be unrolled? - YES
✓Are there two loops operating on the same data that can be combined into one? - YES
✓Are frequently used variables declared register? - YES
✓Are short and commonly called methods declared inline? - YES

Found defects

Identify and describe found defects, opinions and suggestions.

Found defect Id	Package, Class, Method, Line	Defect category	Description

Corrective measures

The data structures used could've been changed as well as the algorithms. Besides that there isn't a comment in every attribute.

Conclusions of the inspection process

Minor and major changes.

Minor: the comments on the attributes;
Major: changing data structures and algorithms, respectively.