Master Thesis

Automatic grades of computer programming assignments

Willmar Rengifo Rengifo

Relatore

Valentina Gatteschi

Thesis
Master's Degree in Computer Engineerig

INFORMATION AND COMMUNICATION
TECHNOLOGIES
POLITECNICO DI TORINO
PIEMONTE, TORINO
DECEMBER 2022

Automatic grades of computer programming assignments

Willmar Rengifo Rengifo

Relatore:

Valentina Gatteschi
Department of Control and Computer Engineering

Abstract

The evaluation method made by professors takes time to evaluate every single student. This thesis is a friendly framework for professors that helps to evaluate a massive student's code using theoretical knowledge of programming languages, best-practices and an approximate way of giving a final score according to the development made by the students. Our goal was to give feedback to the student in a way that they can know where they made the mistakes and for the professor's side, the process can be automated by giving some initial parameters like name of important methods to be evaluated and its own weight. This framework was tested in comparison with real scores from previous exams and the results are similar giving a correct weight on each method and finally but not least, saving time for teachers was the key part of this thesis.

Dedication

Dedicated to my parents, brother and friends.

Acknowledgement

My family are the first people that help me, always were encouraged for reach to my academic goals, without their this doesn't possible.

I would like to thank the closest friends who were for help me with the way was difficult and believe in myself.

Thanks to the professor Valentina Gatteschi that with his patience and guide this thesis is done.

Indice

Elenco delle figure

Capitolo 1

Introduction

Analyzing differences from source code is an important topic that contributes to quality and maintenance, in the course of java programming language is possible implements a procedure of learning based on the differences in the code for assigned a grade, but for the teachers of java programming language spend huge of time checking all java projects of each student that's why the need arose to find useful framework to agile this work.

This thesis was focused on the development of a friendly framework that helps to the evaluation of programming exams and contributes to the feedback for students showing where the differences are, according to the structure given with the instruction by the professor.

Initially, the professor gives the names of the methods that are going to be evaluated and the structure of how the exam will be developed. The second step will be when the student writes the project solution in exam time and the correction of the project is delivered on a second date given by the professor.

The third step is when the professor configure the properties file with the important methods and and their respective weights, to later use the framework to identify the differences between the 2 delivered versions, after that the framework generates a final grades file for each student and generates a file with the counters of the differences found by each method. In this way, students are allowed to see the differences found and this is a way to give some feedback for learning from mistakes. The last step is that the professor can used this evaluation result (grade) generated by the framework to be considered as part of the final grade of each student.

In the first section shows the introduction of the thesis that explain the origin necessity for which this thesis arose. In the second section contains the state of art in which gives an overview of the existing frameworks nowadays for code analysis of Java programming languages.

In the section of Methodology explains the theory of parsing java code in an abstract syntax tree and node management. The Development section talks about the phases of evaluation which are:

1. Navigate parser B which is the parser generated by the correction version delivered.

2. Comparison with parser A which is the parser generated by the exam time version delivered.

3. Upgrade the counters of the differences found between the two projects delivered (exam time and correction time).

4. Evaluation of the grades according to the differences found and the weight given by professor of each method.

5. Stored the counters by method and final grades.

The section results shows the comparison between the grades given by the professor and the grades generated by the framework, the application was proved with several test and his respective properties values and was analyzed which was the best combination of property values that generates grades closest to the grades assigned by the teacher. This thesis was focused on three particular cases, the case one was focused on the type node bodyDeclaration, the second case was based on the type that is not bodyDeclaration and the last one was based on the type node methodDeclaration.

Finally the section conclusion talks about the future works that can be done with this developed framework and highlights the efficient in terms of timing at the evaluation time done by the professors.

Capitolo 2

State of art

This thesis works with different runtime libraries able to take a java source code [7] and create its abstract syntax tree with all information about this source code in the nodes of the tree.

2.1 OOPGrading

- This is a program designed to improve the practices in the course of object oriented program [12] and in the assignment of grades. this tool allows to the professor manager the two versions delivered by the the students of the same project, check the differences between versions and generate a grade as show in the image 3.1.

  These tools implement functionalities of automating testing with JUnit [8] configuration.

- The other important feature that this program is concerned to terms of authentication in the exam thanks to the used Subversion-IDE as eclipse [1], it allows giving a location where the students put their projects. Other features get to thanks to SVN [11] is the support of concurrent development with a Copy-Modify-Merge approach [9].

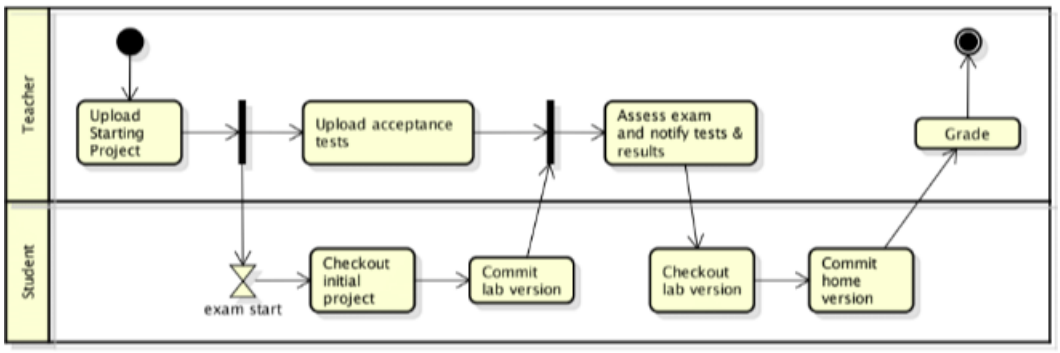

Figura 2.1: Diagram of flow exam.

2.2 CLDIFF

This is a framework proposed by students of the University Fudan in china, the application compare two files, the application is a code differencing between two java file and linked the code differences founds in a efficient summarization util for the recognition of type nodes and assignment of the weigth values.

Initially, the framework has two input arguments for the java files that is request

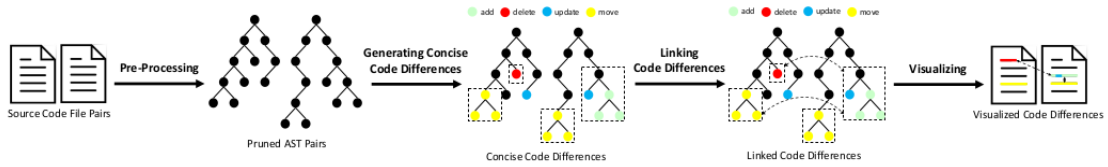

Figura 2.2: An overview of ClDiff [5]

a comparison, the two java files (f a, f b) are representation of a same file but in different times,"file b" is before of changes and "file a" is the same file but after of the changes. The framework cliffdif parsed to an AST's from the source code file java and labeled specified types of node [5].

The framework is able to realize a Hierarchy order in the AST's created. the Hierarchy can be in the subtypes of node as BodyDeclaration, typeDeclaraton, MethodDeclaration, Initializer, FieldDeclaration and EnumDeclaration.
After of the characterization of a nodes of a AST, an edit script is generated and with the framework GumTree [3] the script is used for will be generated fine-grained code differences [5].

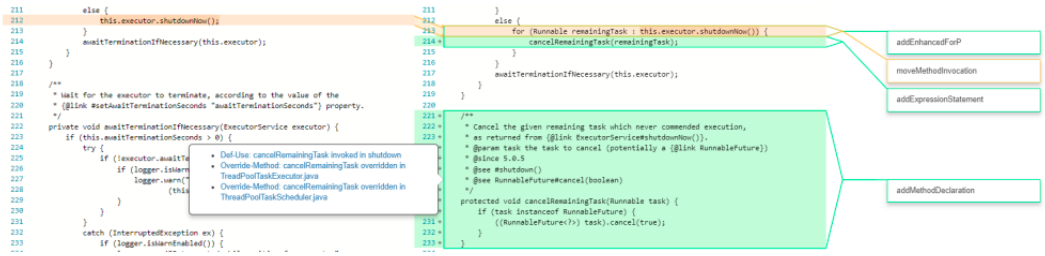

Figura 2.3: View of the framework

2.3 GumTree

GumTree [3] is a framework that use as a based the gumTree Algorithm. This Algorith is in charge of establishing mappings of src and dst and then deducing an edit script.

Exist 3 types of mapping used on GumTree Algorithm such as a greedy top-down algorithm, a bottom-up algorithm or recovery mappings, see figure 2.4

A greedy top-down algorithm is recognized for anchors mappings which means that

first they search for the biggest unmodified pieces of code.

A bottom-up algorithm is called a container mapping which means that they deduce which container of code can be mapped together. A recovery mappings is characterized by searching for additional mappings among their descendants, this means that they look at precise differences in what is leftover in each container [4].

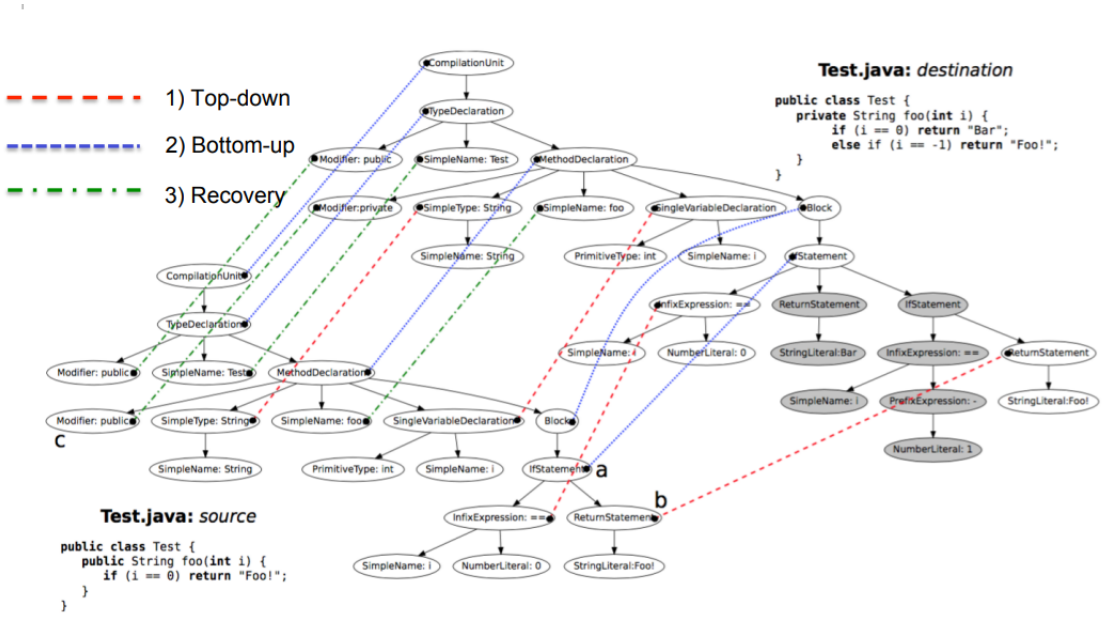

Figura 2.4: Approaches of GumTree framework

This algorithm has 2 main problems, the distance between ordered labeled trees and distance minimum-cost sequence of node edit operations that transform one tree into another. those problems are not efficiency enough for our goal in this thesis.

Capitolo 3

Methodology

3.1 The parser tree

The beginning in the process of understand the differences is in the abstract syntax tree, the convertion from java file to AST [10] is the first step, the AST is the representation used by eclipse for the java source code, each file is represented as a tree of AST nodes, and the nodes are subclasses specialized in a type of java programming language, recognize all nodes and his type is a important task in the process of differentiation in this thesis.

3.2 Java model

A java project is a tree structure thanks to the parser and is similar to a package explorer view, the image show a example of java model 3.1. In this project, the java model is out of scope but is important for future works because it costs less to rebuild a Java model than to rebuild an AST and its nodes are easy handles.

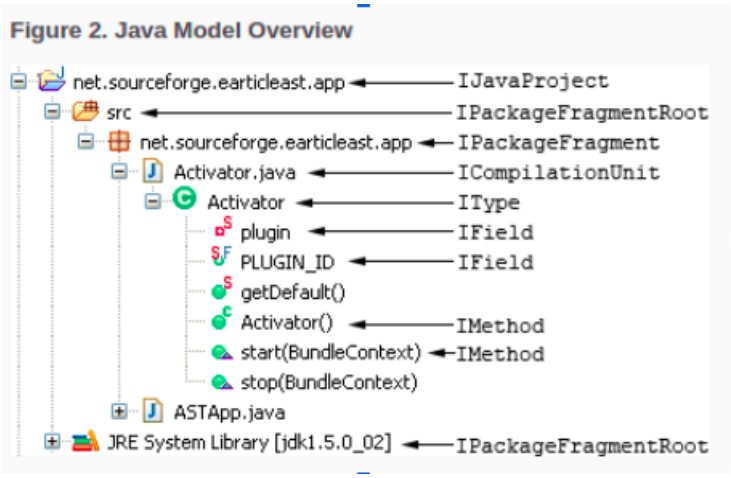

Figura 3.1: Model overview of the project [10].

3.3 Node AST

The following image 3.2 shows two trees created by the framework that are going to be compared between each other in the following step.

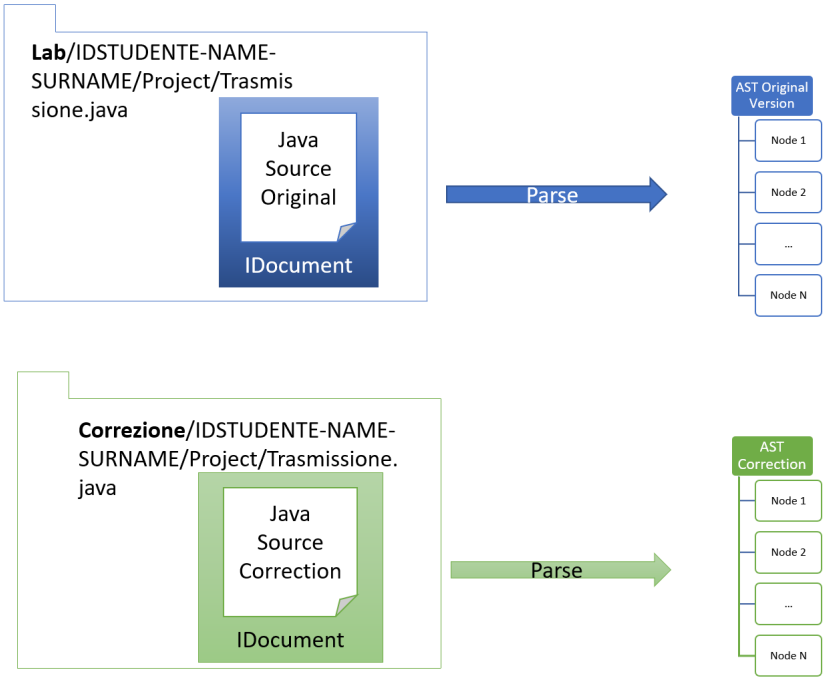

Figura 3.2: Read and write in AST.

The abstract class defines the methods for access to the important features of this type of node, other advantages are that the AST nodes are thread-safe for readers, this is convenient in our application because is not necessary writers, the application does not modify the java source code, only check the type of the node and make to comparison.
There are two ways to reach the values of structural node properties:
Static methods(e.g getName()) or generic, using
getStructuralPropertyDesscriptor(StructuralPropertyDescriptor property).

Each node can represent the following characteristics:

- Types.
- Names.
- Statement- statements.
- Expression - Expresssion
- BodyDeclaration - BodyDeclarations

Exist three kinds of aggrupation for these structural properties as shown in the image3.3, understand this agrupation between children is important in the process of navigate the AST that is done through of recursive calls into the children.:

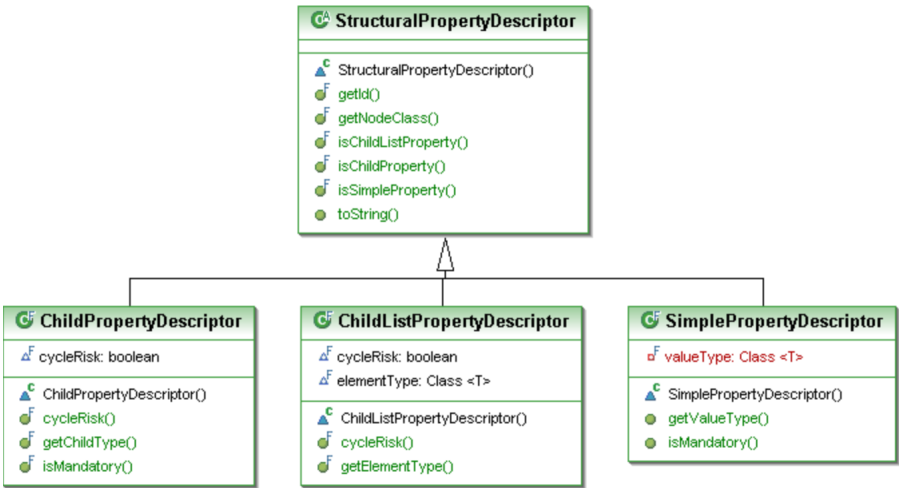

Figura 3.3: Types of children and his properties [2].

The SimplePropertyDescriptor(E.g String, Integer or Boolean), the childProperty-Descriptor is a node that belonging to a subclass of ASTNode and the last one is ChildListPropertyDescriptor which is an array of AST node children of this node. A node AST is the representation of a java code and belongs to a unique AST instance. An AST node has a unique parent node and can have zero or a lot of children, in this way is possible to navigate down from parent to child or upwards from child to parent, In this thesis the navigation through to the abstract syntax tree is realized by means of recursive calls, in the image 3.4 can see a example of AST with differents types of node.

3.4 Visit the parser

3.4.1 CompilationNode

The compilationUnit is the first node, itis the head in the AST and in this thesis is the first node that is sended to the process of comparison. The methods for access the important characteristic into this node, the packageDeclaration through getPackage(), the method imports() return a list of nodes,import declarations, and types() with the list of nodes types declarations. A type declaration node is formed by a class declaration and an interface declaration. The application used the getFields() to return the list declaration or in our java code the global variables and finally, getMethods() returns a list of methods declaration in this file. In the image 3.5, the important children nodes that component the node compilationUnit.

3.4.2 Method declaration node

The documentation of IBM declare that the functions for access to information feature information of a node A of type methodDeclaration are isContructor(), getName()(name of method), parameters() (the list of arguments that must receiver this method), getReturnType2()(return the return type for this method), getBody()(return the body code that belongs to this method). As an example of this we can see the figure 3.6.

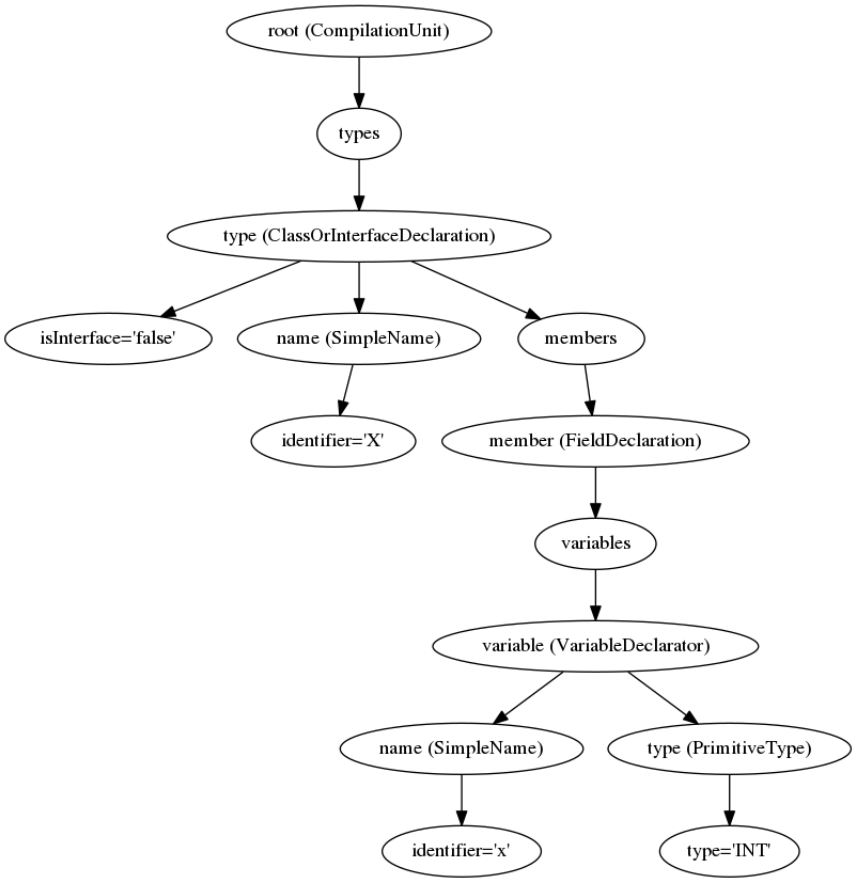

Figura 3.4: Example of grammar in the AST.

```
CompilationUnit:
   [ PackageDeclaration ]
      { ImportDeclaration }
      { TypeDeclaration | EnumDeclaration | AnnotationTypeDeclaration | ; }
```

```
TypeDeclaration:
              ClassDeclaration
              InterfaceDeclaration
ClassDeclaration:
     [ Javadoc ] { ExtendedModifier } class Identifier
                       [ < TypeParameter { , TypeParameter } > ]
                       [ extends Type ]
                       [ implements Type { , Type } ]
                       { { ClassBodyDeclaration | ; } }
```

Figura 3.5: Structure of node CompilationUnit and TypeDeclaration [6].

```
MethodDeclaration:
   [ Javadoc ] { ExtendedModifier } [ < TypeParameter { , TypeParameter } > ] ( Type | void )
       Identifier (
           [ ReceiverParameter , ] [ FormalParameter { , FormalParameter } ]
       ) { Dimension }
       [ throws Type { , Type } ]
       ( Block | ; )
ConstructorDeclaration:
   [ Javadoc ] { ExtendedModifier } [ < TypeParameter { , TypeParameter } > ]
       Identifier (
           [ ReceiverParameter , ] [ FormalParameter { , FormalParameter } ]
       ) { Dimension }
       [ throws Type { , Type } ]
       ( Block | ; )
```

Figura 3.6: Structure of node MethodDeclaration [6].

3.4.3 BodyStatement node

The documentation of IBM advises to use for unparented block node owned by this AST and get the relevant information of the node. All nodes only must be

statements, statements() in which allows to get a list of statements.

The block type AST node is important for be common between others nodes, for example, almost every the nodes that the application evaluate have a part body or the statements. In the image 3.7, the important children nodes that component the node blockStatement.

```
Block:
   { { Statement } }
```

Figura 3.7: Structure of type node Block [6].

3.4.4 TryStatement node

The documentation of IBM advises to use for find the method resources() that return a list of resources of this node try, getBody(), getFinally() and catchClauses are the methods specialized for get the information of this type of node AST. In the image 3.8, the important children nodes that component the node tryStatement.

```
TryStatement:
    try [ ( Resources ) ]
        Block
        [ { CatchClause } ]
        [ finally Block ]
```

Figura 3.8: Structure of type node Try [6].

3.4.5 ThrowStatement node

The documentation of IBM describes the important children nodes that composes the node throwStatement as it is shown in the image 3.9,

```
ThrowStatement:
   throw Expression ;
```

Figura 3.9: Structure of type node Throw [6].

3.4.6 IfStatement node

The documentation of IBM advises to use the method of this library getExpression(), in order to return the expression to check in the ifStatement, which can be a list of expressions. In addition, getThenStatement() return the body of the“then” part and getElseStatement() returns the body of the else part or null if statement ”if” has not. The following image 3.10 shows the important children nodes that component the node IFNode.

```
IfStatement:
   if ( Expression ) Statement [ else Statement]
```

Figura 3.10: Structure type of node If [6].

3.4.7 ForEnhanced node

The documentation of IBM describes the parameters controls are FormalParameter(getBody()), Expression(getExpression()) and SingleVariableDeclaration(getParameter()). The following image 3.11, represent the important children nodes that component the node ForEnhanced.

```
EnhancedForStatement:
   for ( FormalParameter : Expression )
                        Statement
```

Figura 3.11: Structure of type node ForEnhanced.

3.4.8 ForStatement node

The documentation of IBM describes the class that contains the methods for accessing 4 values of for Node, the getBody() that returns the body of this for statement, getExpression() returns the condition expression or null if there is none, inititializers() returns a list of initializer expressions, updaters() returns a list of update expressions. The following image 3.12 shows the important children nodes that component the node forStatement.

```
ForStatement:
   for (
                        [ ForInit ];
                        [ Expression ] ;
                        [ ForUpdate ] )
                        Statement
ForInit:
                Expression { , Expression }
ForUpdate:
                Expression { , Expression }
```

Figura 3.12: Structure of type node For [6].

3.4.9 DoStatement node

The documentation of IBM describes the application of using getBody() for get the Body Statement of this node and the expression in the final while is returned with getExpression(). In the image 3.13, evidence the important children nodes that component the node doStatement.

```
DoStatement:
   do Statement while ( Expression ) ;
```

Figura 3.13: Structure of type node Do [6].

3.4.10 WhileStatement node

Similar to DoStatement, the documentation of IBM describes the application use getBody() for get the Body Statement of this node and the expression is returned with getExpression(). In the image 3.14, evidence the important children nodes that component the node whileStatement.

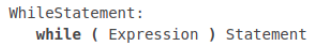

Figura 3.14: Structure of type node While [6].

3.4.11 SwitchStatement node

The documentation of IBM describes the method getExpression() which is in charge of return the expression to evaluate in the switch node and the list of statements() or switch case.In the image 3.15, the important children nodes that component the node switchStatement.

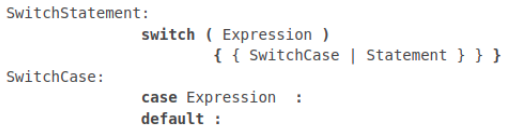

Figura 3.15: Structure of type node Swithc [6].

Capitolo 4

Development

The best way to have an high overview of this thesis is looking at it as simple as possible, throw this whole section will be describe with details each simple step mentioned on this figure 4.1.

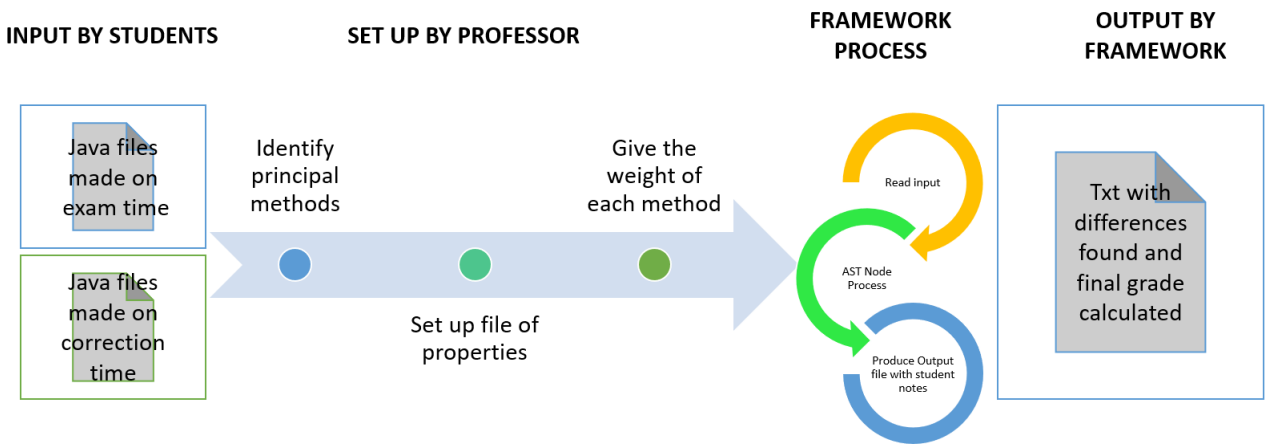

Figura 4.1: General high model overview of the project

This image shows in the left side the java files belonging to the same project but with different versions, version exam or first delivered and version correction, this projects are inputs for the system and this phase is called the input of students. In the other hand is the process of setup the weight values, this process is realized by the professor, the weight values are assigned by types of node and the names of important methods, second criteria of professor, the setup values are load in the ASTExtractorCalifica.Properties file. The next phase is the execution of framework process, in this phase are the subsections of parser of two AST, identification of nodes AST, Comparison between nodes in differents AST and finally calculation of grades. In the final process, the values generated are stored in convenient TXT and excel files, this phase is called the output of framework.

4.1 The framework used ASTExtractor: Abstract syntaxTree

I have started my work with the library abstract syntax tree, widely used in activities such as refactoring, quick fixes, and quick assist. It is an extractor that transforms the java source into a tree form then the tree is more convenient and reliable to analyze and modify programmatically than the text-based source. Firstly, it is necessary that the user provides some code to parser, there are two options, the absolute address filename or the address of the directory which is the project that the user wants to parser. After one or more syntax trees will be created syntax tree for one file found in the project.

The commands for executing the library, only one option is possible

As project:
java -jar ASTExtractor.jar -project="path/to/project"
-properties="path/to/propertiesfile" -repr=XML—JSON
As file:
java -jar ASTExtractor.jar -file="path/to/file"
-properties="path/to/propertiesfile" -repr=XML—JSON

The link where is available: https://github.com/thdiaman/ASTExtractor The goal of the first part of the project is to compare the same java code but in two different phases of development to understand all changes make for the student in the second version with respectively the first version. The second goal is assigned to a grade from the node type that belongs to the change.

4.2 Input Arguments

The first change made to the library ASTExtractor was to add an extra parameter to the arguments so there are two new arguments with respect to the original library, the first new argument is argument is the path name where is located the folder that contains all projects that students delivered and the second argument is the path address to the folder that contains all projects of the students delivered in the first deadline or "made in the time of the exam", in other words, the first argument content is the same project with the corrections made after of the exam then this first argument content should not have many changes. The following image 4.2 shows the diagram of classes of ASTEXTRACTOR.

4.2.1 Folders of the projects

As its shown on 4.3, FolderFirstDelivered="adressLocationOfFolderFirstDelivered" is the address of the folders where the projects students in the second instance of delivery are put. These projects contain changes or corrections made to the first delivery. FolderSecondDelivered="adressLocationOfFolderSecondDelivered" the addresses of the folder where the student's projects are put. All student's projects are in the initial version or the first delivered. The version of the project done in

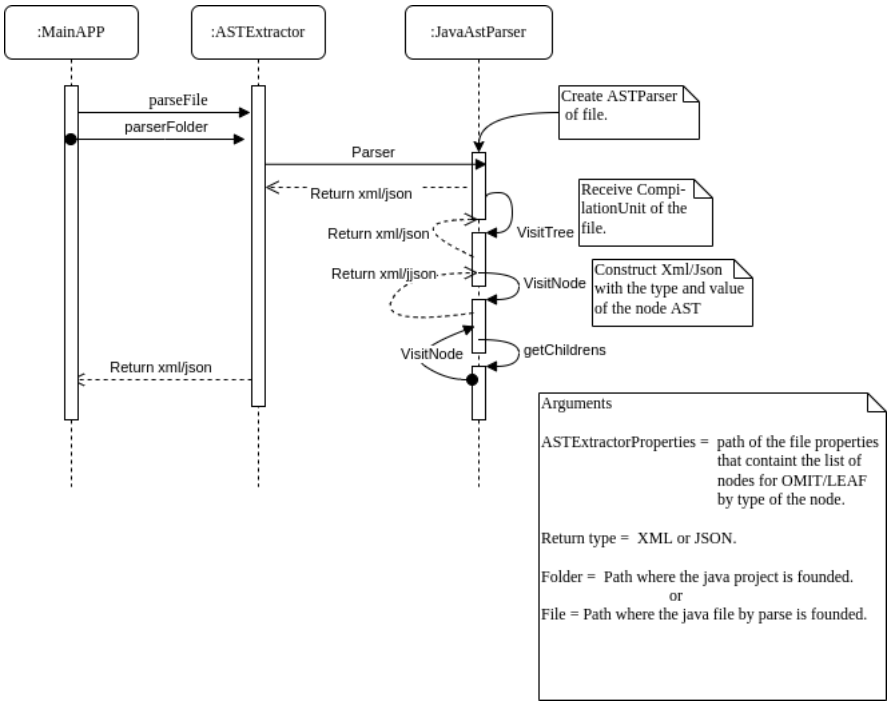

Figura 4.2: Diagram of class of ASTEXTRACTOR.

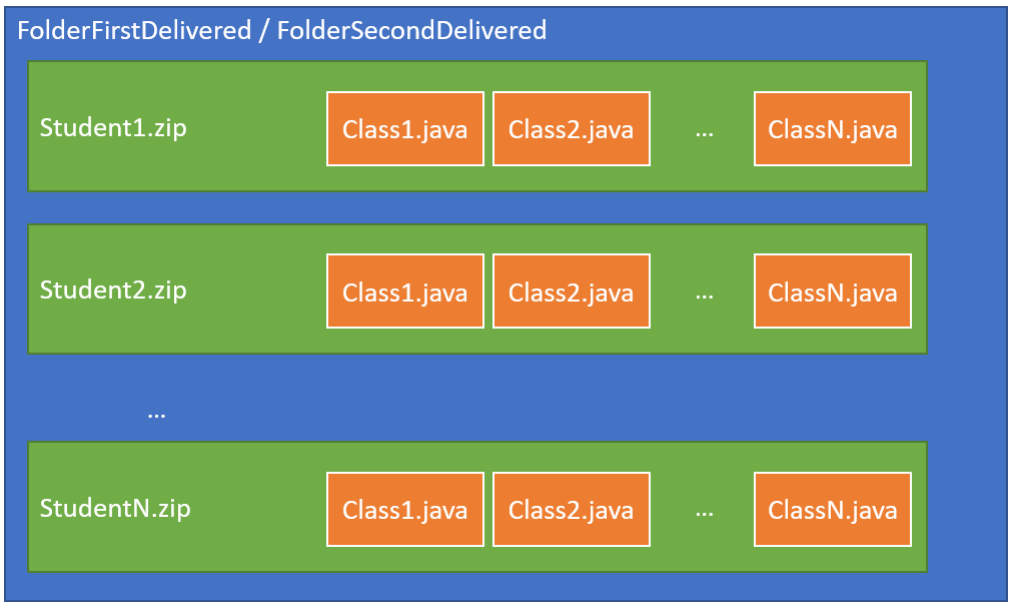

Figura 4.3: Structure of students projects folder.

exam time.

The students are not constrained to deliver the correction of the exam so in this case, the program doesn't assign scores to this student. It is mandatory that the student keep the same structure and the same names for both the first delivered project (the correction) and the second delivered project, on the contrary, the comparison is not possible and the probability of finding differences between the two versions is high and so the final result is considerably affected.

The other new input argument is the evaluation=ASTExtractorCalifica.properties.

This file is for assigning the weight-values that will be stored in the system and used to get the grades for each student and depending on his change type. In the structure of this file, there are separated by the character(%%%%%%%), and the first part is used to specify the names of the classes.

The other part is used to specify the important structures of the nodes. Obviously, all part that is set up in this file takes a special value that affects the final grade of the specified student, as its shown on figure 4.4.

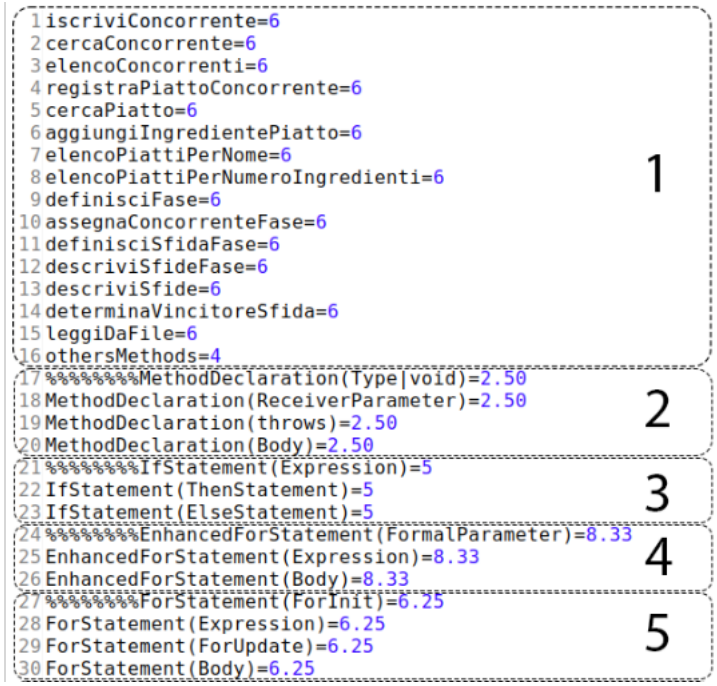

Figura 4.4: The properties file.

4.2.2 The weight properties

The properties file ASTExtractorCalifica.properties contains the values of weight that influence directly in the calculation of the final grade. Those Values are available to be change by the professor criteria, see figure 4.5.

Each section of the properties file is separated by The first section of the file is the name of methods that are considered more important for influence in the evaluation.

From the second section to the final section are the weight that are divided by type of node, in this part are assigned a weight to specific feature for determinate node. 4.6.

This is orden that followed the sections:

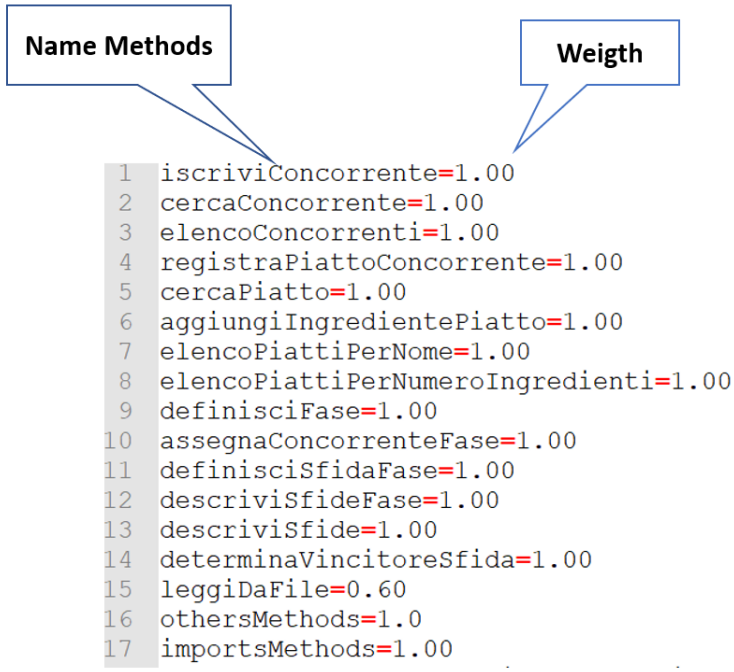

Figura 4.5: Weight by method name. The setup of weight values for the methods to evaluate.

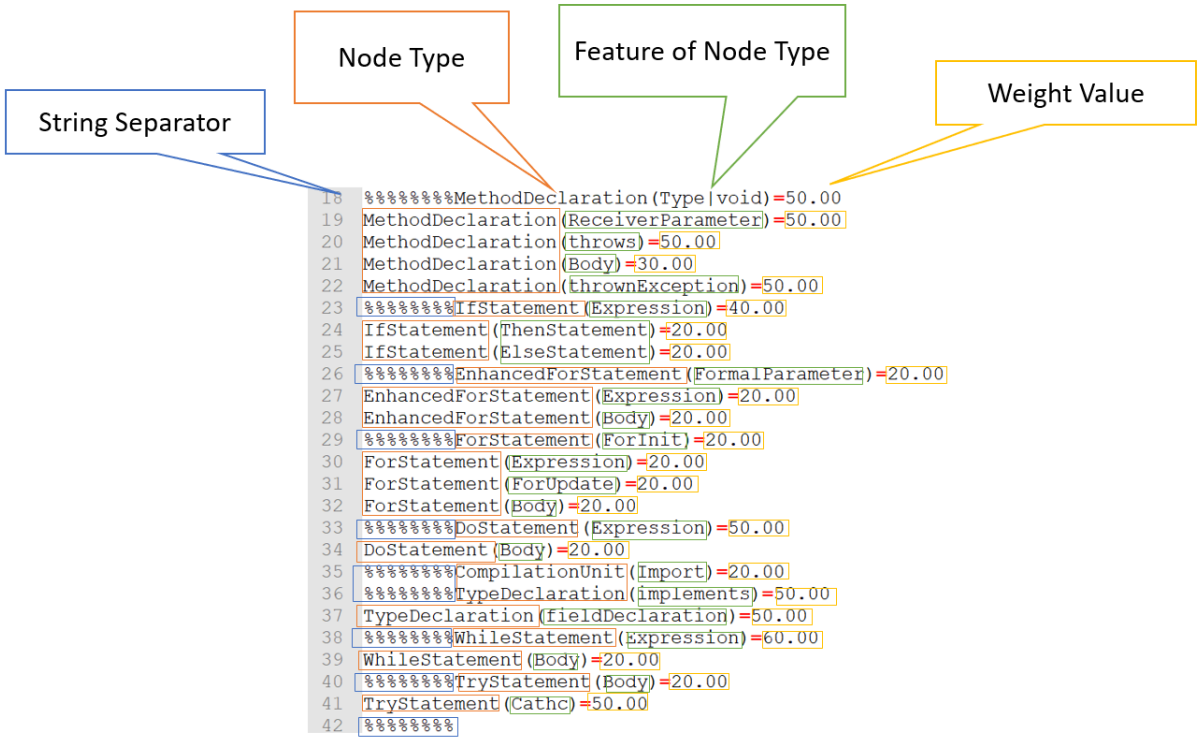

Figura 4.6: Weight by type of node.

1)Name of methods 2)Method Declaration 3)ifStatement 4)Forstatement 5)EnhancedForStatement 6)DoStatement 7)WhileStatement 9)CompilationUnit 10)TypeDeclaration 11)TryStatement

The methods in the list properties are used for the writing of the excel file, which shows the final grades of the students and the counters by methods of the number of differences, one example is the figure 4.5.

The execution -firstVersion=exam//2018-2019//correzione -secondVersion=exam//2018-2019//lab -properties=ASTExtractor.properties -repr=XML -Evaluation=ASTExtractorCalifica.properties.

4.3 Path the FolderFirstDelivered and FolderSecondDelivered

The parserFolder function is the class in charge of listing all java projects in the folderFirstDelivered, one folder for each student, at this point the object student is created, and the system checks if exist the second project version for the same student into the FolderSecondDelivered folder after the projectFirstVersion and projectSecondVersion values are set up in the object student.

projectFirstVersion = Second version or correction version of the same student project(located in FolderSecondDelivered).
projectSecondVersion = The project delivered by the student in exam time(located in FolderFirstDelivered).

4.4 The student class

This class is called only if two similar projects (first version and correct version) are found belonging to the same student. The object student is created before analyzing the files of the java project. This object stored important values about the student, the differences found in the process of comparison of the two versions, the current and final grades, and counters by methods with the number of differences found by methods for this student. The image shows the object student 4.7.

```
public class StudentStatistics {
    public String id;
    public String name;
    public String lastName;
    public String[] methodUsed;
    public double gradeTotal;
    public String folderNameA;
    public String folderNameB;
    public String fileResult;
    FileWriter myWriter;
    static HashSet<String>; methods1 = new HashSet<String>();
    static HashMap<String, integer> coungterPerMethods;
…
}
```

Figura 4.7: The object student.

4.5 Matches of files

With the object student set up, the next step is a new iteration, one for each file found in the projectFirstVersion, in this iteration the operation of matches with the

file that should exist in the project version correction. Only if exist matches between the first and second versions of filenames, the parser of the files and comparison will be done else this eventuality will be written in a txt file and signal as a file not found with the filename that generates this fail.

4.6 Parsing of source code.

For each iteration by file found in the java project student, two abstract syntax tree will be created, the parserA and the parserB, belonging respectively to the files matches, one file in the project delivered in exam time(versionB) and the file in the project delivered in correction time(versionA).

The setup of the handles parsing of java and the extraction of their abstract syntax tree is done in this part, the first node is the belonging to subclass compilationUnit, The following image 4.8 shows the parameters that must be configured before the conversion of source code to AST.

```
121    public static void parseMethod(StudentStatistics studentProject, String fileFirstDelivery,
122                                   String fileCorrectDelivery) {
123        ASTParser parserFirstDelivery = ASTParser.newParser(AST.JLS14);
124        ASTParser parserCorrectDelivery = ASTParser.newParser(AST.JLS14);
125        parserFirstDelivery.setSource(fileFirstDelivery.toCharArray());
126        parserCorrectDelivery.setSource(fileCorrectDelivery.toCharArray());
127        parserFirstDelivery.setKind(ASTParser.K_COMPILATION_UNIT);
128        parserCorrectDelivery.setKind(ASTParser.K_COMPILATION_UNIT);
129        final CompilationUnit cuFirstDelivery = (CompilationUnit) parserFirstDelivery.createAST(null);
130        final CompilationUnit cuCorrectDelivery = (CompilationUnit) parserCorrectDelivery.createAST(null);
131        nodeB.setNode(cuCorrectDelivery);
132        methods.clear();
133        methodsFile.clear();
134        visitFillMethod(cuFirstDelivery);
135        visitNode(cuFirstDelivery,studentProject);
136    }
```

Figura 4.8: Set up handles of parser.

ASTParser.newParser(AST.JLS3); ASTParser.newParser(AST.JLS3); specified the java language specification, in this example JLS3, with the new syntax, differentiation between JLS2 and JLS3 is necessary for terms of compatibility.

parser.setKind(ASTParser.K_COMPILATION_UNIT); Determinates the type of input, there are five kinds of inputs, the ICompilationUnit is one of the inputs, it is the pointer to the java file source.
Entire source file = K_COMPILATION_UNIT
The portion of java code: K_EXPRESSION, K_STATEMENTS,
K_CLASS_BODY_DECLARATION
In this application always is used the option of the entire source file.

The initially navigate into an AST is done in the parserA thanks to the function visit node shown in the image 4.9 in which implements recursive calls for navigate bottom down (the first project version delivered at correction time), all nodes will be asked about their type of node, in this process, the search consists in found specific types of nodes in the AST parserA that are important for the criteria of evaluation, these important nodes are defined by the user in the file ASTExtractorCalifica.properties first of execution.

In the visit to the parserA, each node asks for the instance subclass that belongs, this is the process of recognition, the function that does the recognition of type node is shown in figure 4.10 , which is important because determines the visit abstract subclass that will be called. As say first the abstract subclass has all access methods for this specific type of node and then access to the values of this node.

```
VisitNode(ASTNode node){
                              .

                        /*
                        THE IDENTIFICATION PART OF THE CURRENT NODE
                        if (nodeB instanceof MethodDeclaration) {

                        }else
                                .
                                .
                                .
                        }
                        .
                        .
                        .
                        */


        //NAVIGATE PART THROUGH CHILDREN IN THE TREE
        List<StructuralPropertyDescriptor> props = node.structuralPropertiesForType();
        for (StructuralPropertyDescriptor property : props) {
                Object child = nodeAnew.getStructuralProperty(property);
                if (property.isChildProperty()) {
                        visitNode( (ASTNode) child); //RECURSIVE CALL FOR NAVIGATE THE CHILDREN
                } else if (property.isChildListProperty()) {
                        Iterator<ASTNode> childrenList = ((Iterable<ASTNode>) child).iterator();
                        while (childrenList.hasNext()) {
                                visitNode((ASTNode) childrenList.next()); //RECURSIVE CALL FOR NAVIGATE THE CHILDREN
                        }
                }else if (property.isSimpleProperty()) {

                }
        }
}
```

Figura 4.9: Function for navigating the AST through his nodes.

```
        //IDENTIFICATION OF THE TYPE OF NODE AST
        //THE NODE IS REDIRECTION TO THE CLASS
        //SPECIALIZED IN EXTRACT TO RELEVANT
        //INFORMATION


        if (nodeB instanceof MethodDeclaration) {
                VisitNode.visit((IfStatement) node);
        }else if (nodeB instanceof IfStatement ) {
                VisitIFNode.visit((IfStatement) node);
        }else if (nodeB instanceof DoStatement) {
                VisitDoNode.visit((DoStatement) node);
        }else if (nodeB instanceof EnhancedForStatement ) {
                VisitForEnhanced.visit((EnhancedForStatement) node);
        }else if (nodeB instanceof ForStatement ) {
                VisitForNode.visit((ForStatement) node);
        }else if (nodeB instanceof WhileStatement) {
                VisitWhileNode.visit((WhileStatement) node);
        }else if (nodeB instanceof SwitchStatement) {
                VisitSwitchStatement.visit((SwitchStatement) node);
        }else if (nodeB instanceof TryStatement) {
                VisitTryNode.visit((TryStatement) node);
        }else if (nodeB instanceof ThrowStatement) {
                VisitThrowNode.visit((ThrowStatement) node);
        }
```

Figura 4.10: Process of recognition.

4.7 Visit the parserA

In the next step, all classes that manage the visit to each type of node, add a label that signals the type of node and if the process of comparison with the parserB is false, the label will be used for matches with the value of weight setup in the file ASTExtractorCalifica.properties for this specific type of node.

After the process of identification of a node in the parserA, the process of comparison is called, as is shown on figure 4.31 and the comparison process return to boolean value which means if exists a change in this node but in the versionB.

4.7.1 VisitCompilationNode.java

This class manages the access to the packageDeclaration through getPackage(), the method imports() return a list of nodes import declarations and types() with the list of nodes types declarations.

A type declaration node is formed by a class declaration and an interface declaration. the application used the getFields() to return the list declaration or in our java code the global variables and finally, getMethods() returns a list of methods declaration in this file. In the image 4.11, the important children nodes that component the node compilationUnit.

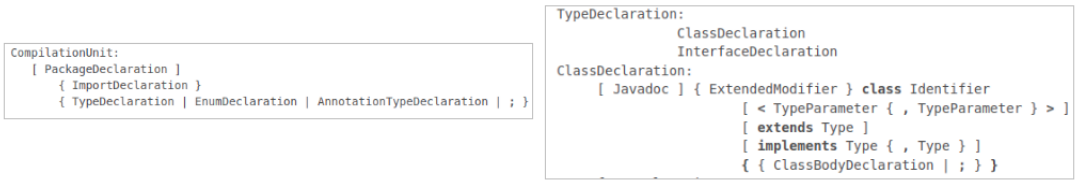

Figura 4.11: Structure of node CompilationUnit and TypeDeclaration [6].

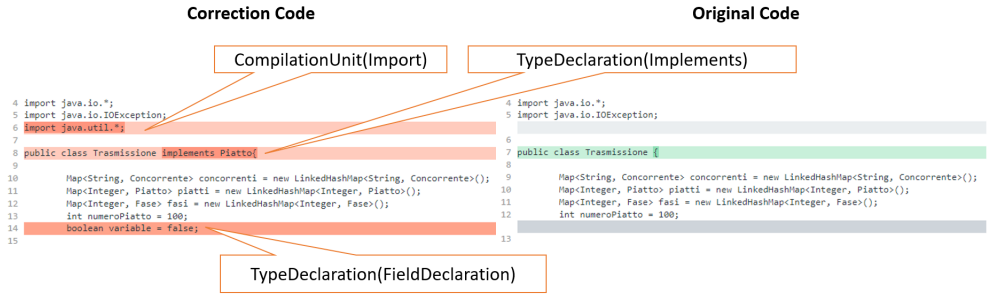

Figura 4.12: Visit compilation node diff.

4.7.2 VisitMethodDeclarationNode.java

In this class are the functions specialized in the access the nodes recognize as a method declaration type, too access the constructor declaration. In the image 4.14, the important children nodes that component the node methodDeclaration.

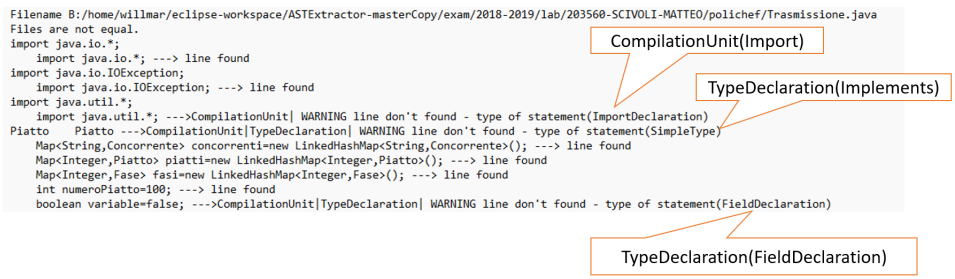

Figura 4.13: Visit compilation node framework overview.

```
MethodDeclaration:
   [ Javadoc ] { ExtendedModifier } [ < TypeParameter { , TypeParameter } > ] ( Type | void )
       Identifier (
           [ ReceiverParameter , ] [ FormalParameter { , FormalParameter } ]
       ) { Dimension }
       [ throws Type { , Type } ]
       ( Block | ; )
ConstructorDeclaration:
   [ Javadoc ] { ExtendedModifier } [ < TypeParameter { , TypeParameter } > ]
       Identifier (
           [ ReceiverParameter , ] [ FormalParameter { , FormalParameter } ]
       ) { Dimension }
       [ throws Type { , Type } ]
       ( Block | ; )
```

Figura 4.14: Structure of node MethodDeclaration [6].

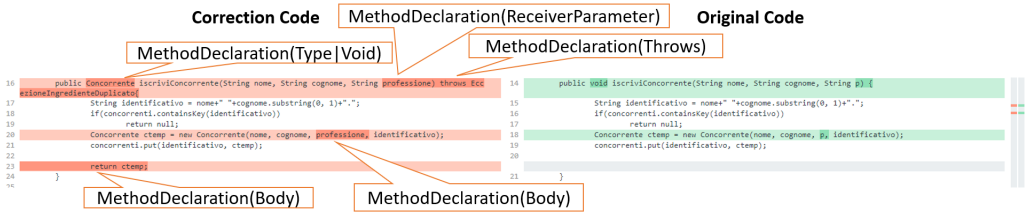

Figura 4.15: MethodDeclaration Diff.

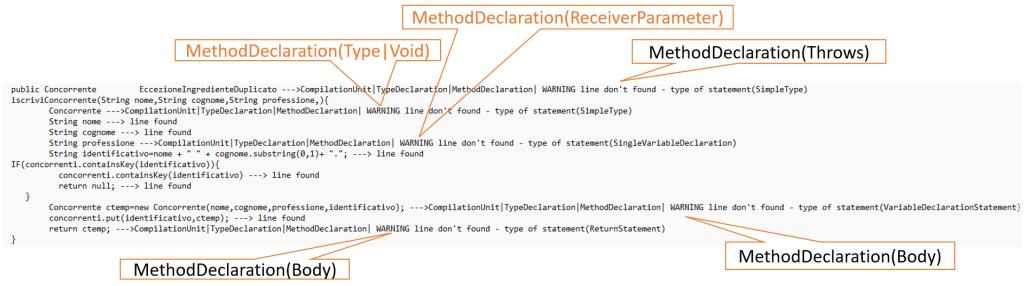

Figura 4.16: MethodDeclaration framework overview.

This is an example of setup in the properties file ASTExtractorCalifica.properties of how the values of weight are set up, the values of weight are applied in dependencies of the site where the change between the two versions is detected. In the image 4.17, a example as how set up the values of weight in dependence of part of code that the professor wish evaluate.

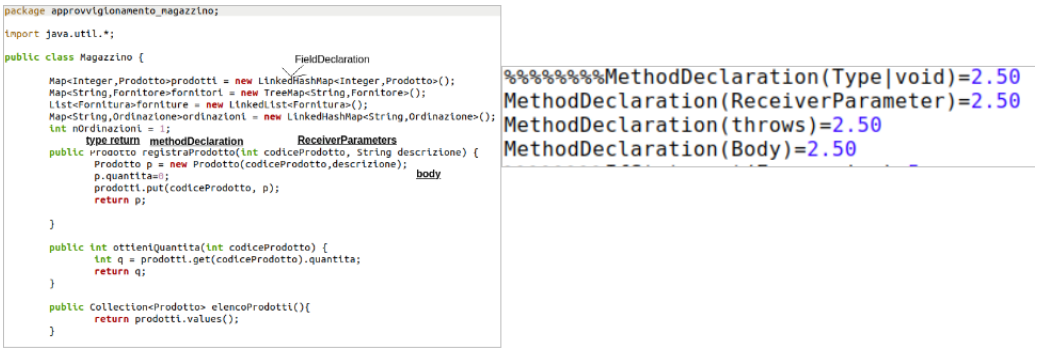

Figura 4.17: The weights values for the node MethodDeclaration.

The methods used for access to information of a node A of type methodDeclaration are isContructor(), getName()(name of method), parameters() (the list of arguments that must receiver this method), getReturnType2() (return the return type for this method), getBody()(return the body code that belongs to this method).

4.7.3 VisitIFNode.java

In this class is possible to find the three more used functions for access to the information that is more relevant for the criteria of evaluation, getExpression(), returns the expression to check in the ifStatement, which can be a list of expressions. getThenStatement() return the body of the"then" part and getElseStatement() returns the body of the else part or null if statement "if" has not. In the image 4.18, the important children nodes that component the node IFNode.

```
IfStatement:
   if ( Expression ) Statement [ else Statement]
```

Figura 4.18: Structure type of node If [6].

Figura 4.19: If declaration.

Figura 4.20: If declaration framework overview.

4.7.4 VisitForEnhanced.java

The parameters controls are FormalParameter(getBody()), Expression(getExpression()) and SingleVariableDeclaration(getParameter()). In the image 4.21, the important children nodes that component the node ForEnhanced.

```
EnhancedForStatement:
   for ( FormalParameter : Expression )
                           Statement
```

Figura 4.21: Structure of type node Forenhanced [6].

4.7.5 VisitForNode.java

This class contains the methods for accessing 4 values of for Node, the getBody() that returns the body of this for statement, getExpression() returns the condition expression or null if there is none, inititializers() returns a list of initializer expressions, updaters() returns a list of update expressions. In the image 4.22, the important children nodes that component the node forStatement.

```
ForStatement:
   for (
                        [ ForInit ];
                        [ Expression ] ;
                        [ ForUpdate ] )
                        Statement
ForInit:
                Expression { , Expression }
ForUpdate:
                Expression { , Expression }
```

Figura 4.22: Structure of type node For [6].

4.7.6 VisitDoNode.java

In this class, the application use getBody() for get the Body Statement of this node and the expression in the final while is returned with getExpression(). In the image 4.23, the important children nodes that component the node doStatement.

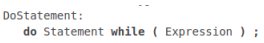

Figura 4.23: Structure of type node Do [6].

4.7.7 VisitWhileNode.java

Similar to visitDoNode.java,the application use getBody() for get the Body Statement of this node and the expression is returned with getExpression(). In the image 4.24, the important children nodes that component the node whileStatement.

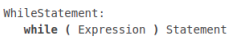

Figura 4.24: Structure of type node While [6].

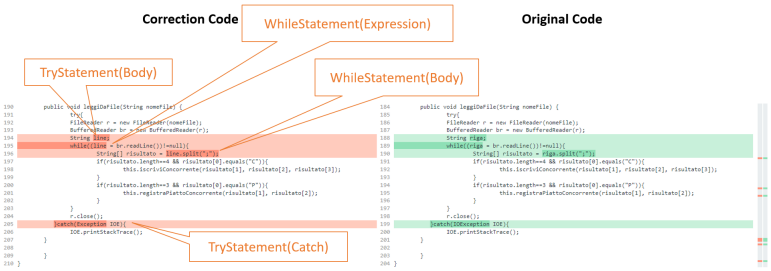

Figura 4.25: While and Try statement.

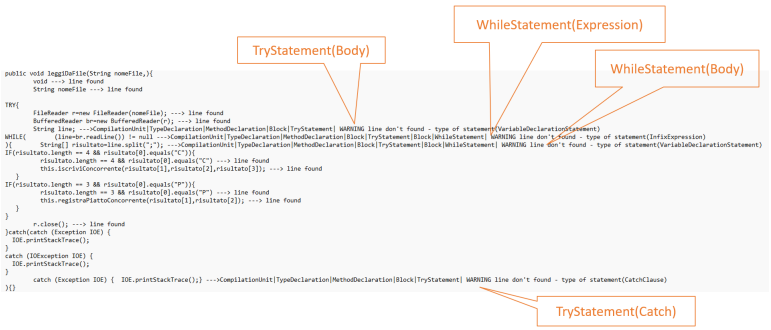

Figura 4.26: While and Try statement framework Overview [6].

4.7.8 VisitSwitchNode.java

getExpression() return the expression to evaluate in the switch node and the list of statements() or switch case. In the image 4.27, the important children nodes that component the node switchStatement.

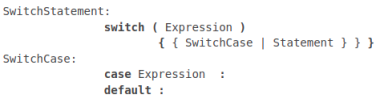

Figura 4.27: Structure of type node Switch [6].

4.7.9 VisitTryStatement.java

In this class can find to the method resources() that return a list of resources of this node try, getBody(), getFinally() and catchClauses are the methods specialized for get the information of this type of node AST. In the image 4.28, the important children nodes that component the node tryStatement.

```
TryStatement:
    try [ ( Resources ) ]
        Block
        [ { CatchClause } ]
        [ finally Block ]
```

Figura 4.28: Structure of type node Try [6].

4.7.10 VisitThrowStatement.java

In the image 4.29, the important children nodes that component the node throwStatement.

```
ThrowStatement:
   throw Expression ;
```

Figura 4.29: Structure of type node Throw [6].

4.7.11 VisitBodyNode.java

This class has only one method for get the relevant information of the node, all nodes only must be statements, statements() allows to get a list of statements, the block type ast node is important for be common between others nodes, for example, almost every the nodes that the application evaluate have a part body or the statements. In the image 4.30, the important children nodes that component the node blockStatement.

```
Block:
   { { Statement } }
```

Figura 4.30: Structure of type node Block [6].

```
nodeB.compareVisitNode(nodeToCheck);
```

Figura 4.31: Call to compare method.

4.8 Compare of statements

The method compareVisitNode is called for each node in the parserA and his duty is to see if exist the same node in parserB.

```
public class compareTypeNodes {
        public static  boolean nodeFound;

        public static void setPropertiesFound(boolean value) {
                nodeFound = value;
        }

        compareVisitNode(ASTNode nodeAForSearch, String typeParent) {
                setPropertiesFound(boolean value);
                visitNodeParserB( nodeBCompilationUnit, "", nodeAForSearch , typeNodeInstanceParent);
                if( nodeFound == true ) {
                        //the nodes of two versions(original and correction)are equals
                        }
                else {
                        //Exist to difference between the two versons, the node identified is store.
                }
        }

        visitNodeParserB(ASTNode nodeB, String nodeTypeB, ASTNode  nodeAForSearch
                , String typeNodeInstanceParentA) {
                //The visit to AST of the parserB
                compareNodes(nodeAForSearch, typeParameterMethodB);
        }

        private static void compareNodes(ASTNode nodeAForSearch, ASTNode x) {
                String nodeTypeA = ASTNode.nodeClassForType(nodeAForSearch.getNodeType()).getSimpleName();
                String nodeTypeB = ASTNode.nodeClassForType(x.getNodeType()).getSimpleName();
                if(nodeTypeA.contentEquals(nodeTypeB)) {
                        if(nodeAForSearch.toString().contentEquals(x.toString()) ) {
                                nodeFound = true;
                        }
                }
        }
}
```

Figura 4.32: The method compares:two nodes

The parameter for sending is the same node that is currently checked, and the value for return is a value boolean that specified if found a similar node in the parserB

VisitNodeParserB The process of comparison is realized into the tree created by the parserB of the project correction, at this point, the application has a node reference in the parserA and starts the navigate into the AST parserB.

The navigation through the AST created for the parserB will check only the nodes that have the same subclass in the type of nodes following the same root in terms of the parents, so only take a root that matches between parents in the AST parserB and the parents of the node in the AST parserA, in this way the navigate through the AST parserB is done more efficiently than in the AST parserA. Started from the initial parent, compilationUnit to the last parent of the node reference in the parserB. The nodes that satisfied this rule are passed to the process of identification, the same process made with the nodeA in parserA explained in the chapter 4.7 is made with these nodes found in the visit to parserB and the nodes identified with the instances that the node reference will be check.

Finally the nodes that passed the last two controls, they will be subject to a final check: With these two final controls, the process of comparison is ended, and the global variable “nodeFound” is set up to false in the init of process comparison of each node in the tree of parseA after starting the process of visit and recognizing

```
VisitNodeParserB(ASTNode node){

        //THE CURRENT NODE ONLY WILL BE PROCESS IF
        //THE PARENTS OF NODE IN PARSERB AND THE
        //PARENTS OF NODE IN PARSERA ARE EQUALS
                .
        if(typeNodeInstanceParentA.contentEquals(nodeTypeB)) {

                /*THE PROCESSS IDENTIFICATION OF THE CURRENT NODE
                        if (nodeB instanceof MethodDeclaration) {

                        }else if (node instanceof IfStatement) {

                        }.
                         .
                         .
                         .
                */
         }
        //NAVIGATE PART THROUGH CHILDREN IN THE TREE PARSERB
        List<StructuralPropertyDescriptor> props = node.structuralPropertiesForType();
        for (StructuralPropertyDescriptor property : props) {
                Object child = nodeAnew.getStructuralProperty(property);
                if (property.isChildProperty()) {
                        visitNode( (ASTNode) child); //RECURSIVE CALL FOR NAVIGATE THE CHILDREN
                } else if (property.isChildListProperty()) {
                        Iterator<ASTNode> childrenList = ((Iterable<ASTNode>) child).iterator();
                        while (childrenList.hasNext()) {
                                visitNode((ASTNode) childrenList.next()); //RECURSIVE CALL FOR NAVIGATE THE CHILDREN
                        }
                }else if (property.isSimpleProperty()) {

                }
```

Figura 4.33: The visit to the parserB

```
if(nodeTypeA.contentEquals(nodeTypeB)) {
        if(nodeAForSearch.toString().contentEquals(x.toString())){
                nodeFound = true;
        }
}
```

Figura 4.34: The nodes are equals?

the node in the parserB that will make the global variable true only if the belonging node to parserB has the following requirements:

- The chain formed by the type of node of his parent is the same in the node references in parserA and the node Found in the process of comparison.
- The nodes parserA and parserB have the same type of nodes.
- The string values that are represented in the code these nodes are the same.

4.9 Store differences by file

For each node in the Ast parserA that active the process of comparison, one of the first steps is setup nodeFound to false, and in the final of comparison this variable global is setup to true if the node has the three rules described in the previous charter otherwise the node of the AST created by parserA is different or not exist the node in the AST created by parserB and therefore will be managed as node different and store into a collection java.

```
public class compareTypeNodes {


	HashMap<int, ASTNode> ChangesByFileMap =new HashMap<int, ASTNode>();
	int counterDifference = 0;

	public static void setListNodesChangesByFile(ASTNode listNodesForSave) {
		counterDifference = counterDifference + 1;
		ChangesByFileMap.put(counterDifference, listNodesForSave);
	}
	HashMap<String, ASTNode> getChangesByFileMap() {
		return ChangesByFileMap;//The main class ask for this collection for each file in the
					//   project after that finish the process of
		}		// the comparison of all nodes in the file versionA

	compareVisitNode(ASTNode nodeAForSearch,  String typeParent) {
		visitNodeParserB(nodeAForSearch);//call to visit the parserB,if nodeA is found
						//in the parserB, the nodeFound will be "true"
		if( nodeFound == true ) {
			//the nodes are equals, Not changes between the nodes in versionA and versionB.
			}
		else { //Exist to difference between the two versons, the node identified is store.
			setListNodesChangesByFile( nodeAForSearch);//Store de node only if
		}					//the process of comparison between
	}					//the node of versionA did not find
}					//his match in versionB
```

Figura 4.35: Store nodes differences found

Finally exist a collection java where is stored all nodes with differences found, this structure is clear always in the init of processing each file, the colecction is to type map collection with a String as a key and a AST node as a value, the IdentificationDiff and the different node AST are the key and the value, respectively. The IdentificationDiff is a key formed by the name method that belong or has the first parent in his familiar'chain and additionally a counter by method stored.

```
HashMap<int, ASTNode> ChangesByFileMap =new HashMap<int, ASTNode>();
```

Figura 4.36: The java collection that store the differences nodes.

4.10 Calculation of the grade

When processing of identification of all nodes in parserA created by a file into the first project delivered and the comparison with parserB created by a homologous file in the version correction are finished, the collection java that stores the differences nodes found can be null or has a one or more nodes AST.

At this point is done the check to all differences generated by the file so the process of calculation of the grade of a student is done in this step for each file. For each file processing the collection that store the node's differences are navigated and each difference activate a process of calculation, the variables String weightMethod and weightType are reached by means of getWeightMethod() and getReference().

The method getWeightMethod() receives the label that indicates the name method that this node is part and returns the value of weight set up in the ASTExtractorCalifica.properties to this method name. the method getWeightReference(), receives

```
//Set up of the methods counters in 0 for each project student
studentProject.setCounterByMethods(methods);
for (File file : list) {//Identification and comparison process of
                //all nodes that have each file.
                .
                .
        //final part of processing a file in the project java
        changesMap = nodeB.getChangesByFileMap();
        Iterator<Entry<int, ASTNode>> it = changesMap.entrySet().iterator();
        //bucle  for process all nodes that generate
        while (it.hasNext()) {//the difference  in the current file.
                String nameMethod = nodeB.nameMethodDifference(nodeChange);//get the method name of his parent
                //get the value of weight related to the name of method.
                double weightMethod = ASTExtractorCalifica.getWeightMethod(nameMethod);
                //get the value of weight related with this type of node.
                double weightReferences = ASTExtractorCalifica.getReferenceWeight(weightTypeWithoutName);

                //the process of substract is call with the two values, for each difference that exist
                // exist in the file, one call to this process of substract is done.
                double weightSubstract = studentProject.minusGradeTotal(weightMethod, weightReferences);
                studentProject.updateCounterByMethods(nameMethod, 1);
        }
}
```

Figura 4.37: The calculation process.

the label that indicates the type of node of the current node different and the method returns a value of weight that indicates the weight assigned to this type of node.
In each iteration of the control of a project, in the init part the student begins with a grade of 30, the most high grade but while the program checks all files into the project for each different node, the values weightMethod and weightReferences generate a new value that will be used in the calculate of the new grade of the student. in the practices, the student has an initial grade in the most high-grade but with each difference found, the subtract operation is applied over this grade minus the value generate for the values of the weight of the specific node that found different. When the analysis of all files in the project of a student has been finished, the current grade of the student is taken as the final grade by the student.

```
minusGradeTotal(double weightMethod, double referencesWeight) {
                this.gradeTotal = this.gradeTotal - (weightMethod*referencesWeight)/100;
                return (weightMethod*referencesWeight)/100;//return value that substract to the totalValue(grade)
}
```

Figura 4.38: The formula.

MinusGradeTotal is the method that realizes a new calculation for each different node so each difference found in the file, decreases the grade of a student, in a percentage defined by the weight values.

The values of weightMethod and referencesWeight are multiplied and the result is divided by 100. Finally, the number result of the last operation is subtracted from the current grade to generate the new grade of the student, which is stored in the variables of the object student.

4.11 Differences founds

The framework generates inside of each student folder the final result txt, which contains the information about the nodes belonging to the file, if the node is found, a print message will be printed ”line found”, othewise the label will be a ”warning” message.

The differences found will be shown at the final check of the file. Each difference found has information about localization inside the code, type of node and weight assigned in the properties file.

Figura 4.39: Differences found txt file.

4.12 Update counters

```
//load the methods set up in the properties
//file ASTExtractorCalifica.properties
methods = parserStudentFileA.getMethods();
//set the counters by methods in 0
studentProject.setCounterByMethods(methods);
for (File file : list) {
                .
                .
        //final part of processing a file in the project java

        //loop of the differences nodes found  in the current file.
        while (it.hasNext()) {
                //get the method name of his parent
                String nameMethod = nodeB.nameMethodDifference(nodeChange);
                //call of method that get the counter value for this
                //method name and update it with plus one to the current value
                studentProject.updateCounterByMethods(nameMethod, 1);
        }
}
```

Figura 4.40: Operation for update counters.

When the analysis of the java project of each student is started, the student object loads the methods set up in the file properties ASTExtractorCalifica.properties and set up these counters with 0 after the process of analyzing each file is initiated. As said before, every time that in each file finished the process of identifying the nodes in parseA and comparison with the nodes in parserB, the next step is to process the collection of nodes differences, that for each node difference the operation calculation of the grade is call, subtracting a percentage of the current grade.

The next step is to update the counters by methods that have the object student, and update the current counter in plus one. the methods set up for each student in the analysis of the project are taken for the properties file ASTExtractorCalifica.properties.

4.13 The outputs

The files generated by the execution of the program are basically a txt in each folder of the student, this is specially practiced for check the differences that caused a discount to the final grade by individual student, the other important file is the excel, this excel file containt the final grade of all student and counters with the differences founds by methods.

4.13.1 The structure result final excel

The final results are written in an excel file but first, the dates are stored in a global structure, which is called "resultGrades" and is a treeMap collection that stores the final result of each student.

```
Map<String, Object[]> resultGrades = new TreeMap<String, Object[]>();
```

Figura 4.41: call to the method that created and write Excel file.

for each project student processed, the final result grade must store in resultGrades, other important information stored is the personal information of the student and all methods specified in the file ASTExtractorCalifica.properties with his respective counters that indicate the number of changes or different founds divided by methods.

```
resultGrades.put(numCadena, new Object[]{
                studentProject.getId(),
                studentProject.getLastName(),
                studentProject.getName(),
                studentProject.getGradeTotalConvertString() ,
                studentProject.getCounter("method 1"),
                studentProject.getCounter("method 2"),
                studentProject.getCounter("method 3"),
                   .//all counters of methods set up in
                   .// ASTExtractorCalifica.properties
                   .
                   .
                   .
                studentProject.getCounter("method n")});
```

Figura 4.42: The collection stored the updates.

Finally, when all projects java are checked, the data in the structure resultGrades are sent to the method CreateExcelNew(). This method receives the structure collec-

```
dataDocuments.CreateExcelNew(resultGrades);
```

Figura 4.43: The method create and write Excel file.

tion and thanks to importing the library org.apache.poi.hssf.usermodel.HSSFWorkbook

is possible to store information about the final result grades for every student. The final excel that is generated by the framework contains the id of the student, name, surname, the final score generated by the framework, the counter of differences found by the method. the following image shows an example of the final excel that we are describing in this section. 4.45

```
public void CreateExcelNew(Map<String, Object[]> datos) {

    Workbook workbook = new HSSFWorkbook();//Create the book of work
    Sheet sheet = workbook.createSheet("Hoja de datos");//new sheet
    //For each line is created a object array(Object[])


    //iteration over data for write in the sheet
    Set<String> keyset = datos.keySet();
    int numeroRenglon = 0;
    for (String key : keyset) {
        Row row = sheet.createRow(numeroRenglon++);
        Object[] arregloObjetos = datos.get(key);
        int numeroCelda = 0;
        for (Object obj : arregloObjetos) {
            Cell cell = row.createCell(numeroCelda++);
            if (obj instanceof String) {
                cell.setCellValue((String) obj);
            } else if (obj instanceof Integer) {
                cell.setCellValue((Integer) obj);
            }
        }
    }
}
```

Figura 4.44: The java collection that store the differences nodes.

ID	Grade	Descriptio	iscriviCon	cercaCon	elencoCo	registraPi	cercaPiat	aggiungiIr	elencoPia	elencoPia	definisciF	assegnaC	definisciS	descriviSt	descriviSt	d
175536	25.360000		1	0	0	2	0	0	0	0	0	0	1	0	7	
191231	29.6		0	0	0	0	0	0	0	0	0	0	0	0	0	
203560	22.240000		5	0	0	0	0	2	0	0	0	0	3	0	3	
204508	5.5200000		0	0	9	2	4	4	3	3	3	3	3	6	7	
206263	17.920000		0	0	0	3	0	0	1	0	0	0	1	1	5	
216642	19.360000		3	0	0	0	0	0	0	0	0	0	4	5	5	
222738	24.000000		1	0	1	0	0	1	0	0	0	0	1	0	6	
224278	30.0		0	0	0	0	0	0	0	0	0	0	0	0	0	
224387	15.280000		0	0	2	6	0	0	0	0	2	2	3	7	5	
224775	29.200000		0	0	0	0	0	0	0	0	0	0	1	0	0	
224872	28.000000		1	0	1	0	0	0	0	0	0	0	0	0	0	
226199	27.920000		0	0	0	0	0	0	0	0	0	0	0	0	0	
226203	21.200000		1	0	0	0	0	0	0	0	0	0	0	5	5	
226500	17.520000		0	0	0	0	0	0	0	0	0	0	4	1	11	

Figura 4.45: Final excel data for exam 2018 - 2019

Capitolo 5

Evaluation

In this section shows the results obtained by the framework applied a real case test from exam of 2018 - 2019, this section was evaluated with the project called "Polichef".

For this section, 125 students has been delivered his projects for evaluation, each student has the first version project and the correction project.The final grades of each student generate by the framework was compared by the grade assigned by the teacher. For the test was prepared three differents ASTExtractorCalifica.properties with different parameters, the test are called: test_ChangesBodyDeclaration, test_ChangesNotBodyDeclaration, test_ChangesMethods;

To all test was calculated a media of the grades assigned by the framework, the best configuration found so far was the second test_ChangesNotBodyDeclaration with 0.70,in others test, the results were 1.30 for the first and 0.90 for the third.

5.1 test_ChangesBodyDeclaration

In this section the image 5.1 represents the weight values that were assigned in the ASTExtractorCalifica.Properties for the first test and the figure 5.2 expose the results applied in the test case one with the properties mentioned before, in this test the classification was made with all values that belong to type node bodyDeclaration with values of 20 and the others types of nodes with values of 50(for example, expressions, return exam, receiverParameters, returnParameter/void etc).

In the tables belonging to this test 5.1 5.2 5.3 the second column is the grades assigned by the framework, the third column is the grade assigned by the teacher, and in the final column is possible to see the difference between the two grades for each student, for this test the media is 1.30, this is the least efficient result test according to the teacher's grade.

5.2 test_ChangesNotBodyDeclaration

The image 5.3 shows the weight values that were assigned for the second test and the figure 5.4 shows the results applied in the test case two with the properties

```
%%%%%%%%MethodDeclaration(Type|void)=50.00
MethodDeclaration(ReceiverParameter)=50.00
MethodDeclaration(throws)=50.00
MethodDeclaration(Body)=30.00
MethodDeclaration(thrownException)=50.00
%%%%%%%%IfStatement(Expression)=40.00
IfStatement(ThenStatement)=20.00
IfStatement(ElseStatement)=20.00
%%%%%%%%EnhancedForStatement(FormalParameter)=20.00
EnhancedForStatement(Expression)=20.00
EnhancedForStatement(Body)=20.00
%%%%%%%%ForStatement(ForInit)=20.00
ForStatement(Expression)=20.00
ForStatement(ForUpdate)=20.00
ForStatement(Body)=20.00
%%%%%%%%DoStatement(Expression)=50.00
DoStatement(Body)=20.00
%%%%%%%%CompilationUnit(Import)=20.00
%%%%%%%%TypeDeclaration(implements)=50.00
TypeDeclaration(fieldDeclaration)=50.00
%%%%%%%%WhileStatement(Expression)=60.00
WhileStatement(Body)=20.00
%%%%%%%%TryStatement(Body)=20.00
TryStatement(Cathc)=50.00
%%%%%%%%
MethodDeclaration|IfStatement|ForStatement=0.02
MethodDeclaration|IfStatement=0.03
CompilationUnit|TypeDeclaration|MethodDeclaration|Block|ReturnStatement=0.09
default=0.01
%%%%%%%%
```

Figura 5.1: File Properties Case 1

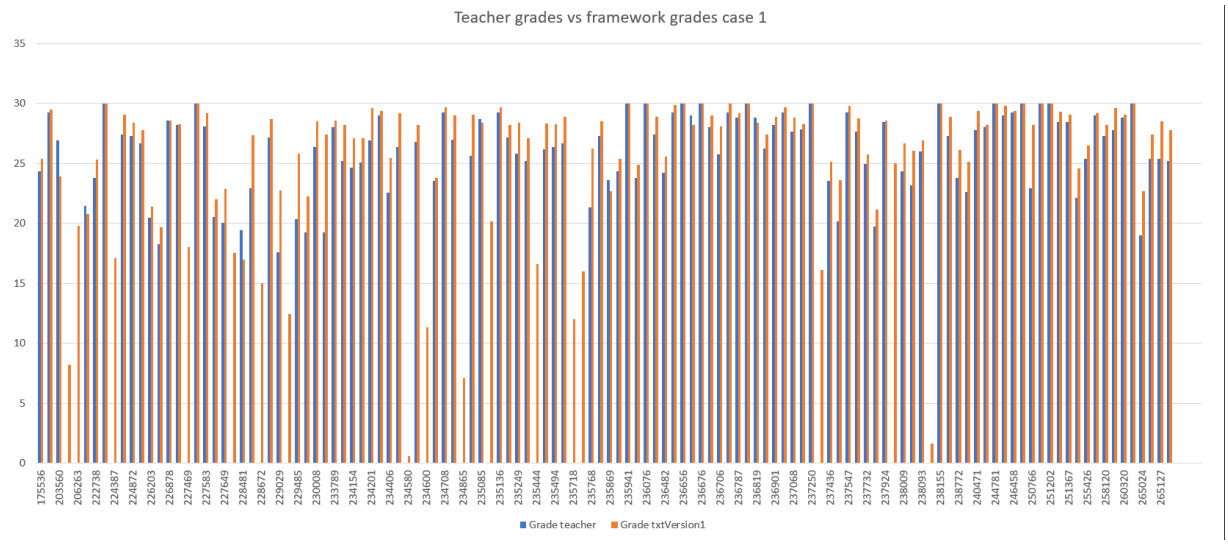

Figura 5.2: Changes body declaration, case 1

previous mentioned, in this test the classification was made with all values that belong to type node bodyDeclaration with values of 20 and the others types of nodes with values of 40(for example, expressions, return exam, receiverParameters, returnParameter/void etc). In the tables belonging to this test 5.4 5.6 5.5 the second column is the grades assigned by the framework, the third column is the grade assigned by the teacher, and in the final column is possible to see the difference between the two grades for each student, for this test the media is 0.70, this is the

best media that represent the most accurate test.

```
%%%%%%%%MethodDeclaration(Type|void)=80.00
MethodDeclaration(ReceiverParameter)=80.00
MethodDeclaration(throws)=80.00
MethodDeclaration(Body)=30.00
MethodDeclaration(thrownException)=80.00
%%%%%%%%IfStatement(Expression)=80.00
IfStatement(ThenStatement)=20.00
IfStatement(ElseStatement)=20.00
%%%%%%%%EnhancedForStatement(FormalParameter)=80.00
EnhancedForStatement(Expression)=80.00
EnhancedForStatement(Body)=20.00
%%%%%%%%ForStatement(ForInit)=40.00
ForStatement(Expression)=40.00
ForStatement(ForUpdate)=40.00
ForStatement(Body)=20.00
%%%%%%%%DoStatement(Expression)=40.00
DoStatement(Body)=40.00
%%%%%%%%CompilationUnit(Import)=80.00
%%%%%%%%TypeDeclaration(implements)=80.00
TypeDeclaration(fieldDeclaration)=80.00
%%%%%%%%WhileStatement(Expression)=80.00
WhileStatement(Body)=20.00
%%%%%%%%TryStatement(Body)=20.00
TryStatement(Cathc)=50.00
%%%%%%%%
MethodDeclaration|IfStatement|ForStatement=0.02
MethodDeclaration|IfStatement=0.03
CompilationUnit|TypeDeclaration|MethodDeclaration|Block|ReturnStatement=0.09
default=0.01
%%%%%%%%
```

Figura 5.3: File Properties Case 2

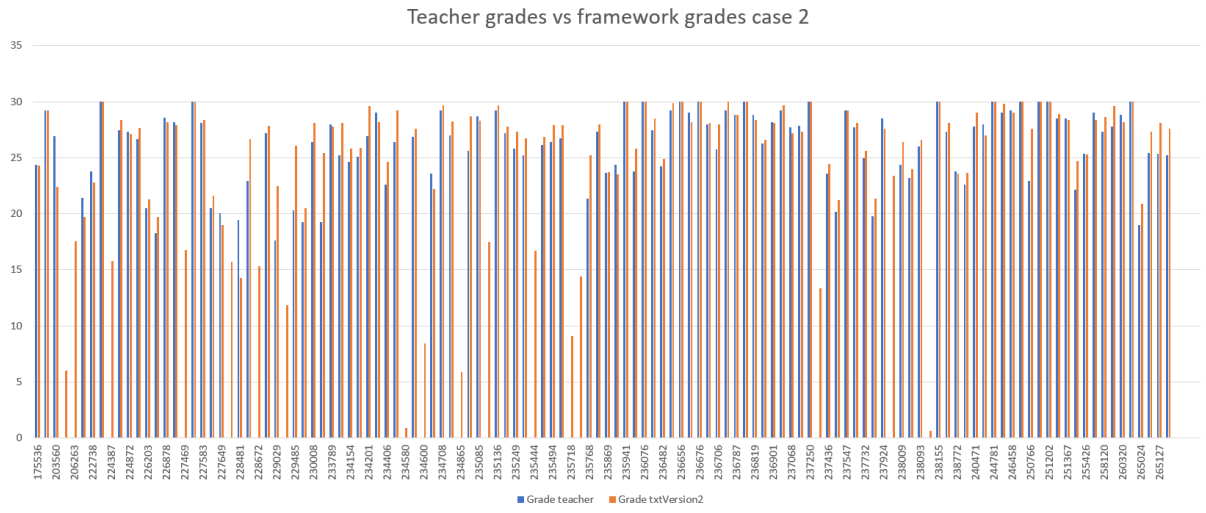

Figura 5.4: Changes not body declaration, case 2

5.3 test_ChangesMethodDeclaration

This image 5.5 expose the weight values that were assigned for the first test and the figure 5.6 expose the results applied in the test case three with the properties mentioned before, in this test the classification was made with all values that belong to type node bodyDeclaration with values of 40 and the others types of nodes with values of 40(for example, expressions, return exam, receiverParameters, returnParameter/void etc). In the tables belonging to this test teacher 5.7 5.8 5.9 the second column is the grades assigned by the framework, the third column is the grade assigned by the teacher, and in the final column is possible to see the difference between the two grades for each student, for this test the media produce by the results is 0.90, this is the second best properties values.

```
%%%%%%%%MethodDeclaration(Type|void)=40.00
MethodDeclaration(ReceiverParameter)=40.00
MethodDeclaration(throws)=40.00
MethodDeclaration(Body)=40.00
MethodDeclaration(thrownException)=40.00
%%%%%%%%IfStatement(Expression)=40.00
IfStatement(ThenStatement)=40.00
IfStatement(ElseStatement)=40.00
%%%%%%%%EnhancedForStatement(FormalParameter)=40.00
EnhancedForStatement(Expression)=40.00
EnhancedForStatement(Body)=40.00
%%%%%%%%ForStatement(ForInit)=40.00
ForStatement(Expression)=40.00
ForStatement(ForUpdate)=40.00
ForStatement(Body)=40.00
%%%%%%%%DoStatement(Expression)=40.00
DoStatement(Body)=40.00
%%%%%%%%CompilationUnit(Import)=40.00
%%%%%%%%TypeDeclaration(implements)=40.00
TypeDeclaration(fieldDeclaration)=40.00
%%%%%%%%WhileStatement(Expression)=40.00
WhileStatement(Body)=40.00
%%%%%%%%TryStatement(Body)=40.00
TryStatement(Cathc)=40.00
%%%%%%%%
MethodDeclaration|IfStatement|ForStatement=0.02
MethodDeclaration|IfStatement=0.03
CompilationUnit|TypeDeclaration|MethodDeclaration|Block|ReturnStatement=0.09
default=0.01
%%%%%%%%
```

Figura 5.5: File Properties Case 3

5.4 test_Overall

The images show the line graph between the grades generated by the framework in each case and the teacher's grades figure 5.7. Notice that those students that have INS grades cause wired cases.

The images show the line graph between the grades generated by the framework in each case and the teacher's grades figure 5.8. In this representation was not include the student with INS grades, and doing in this way we appreciated accurate results, almost the same as teacher' grade.

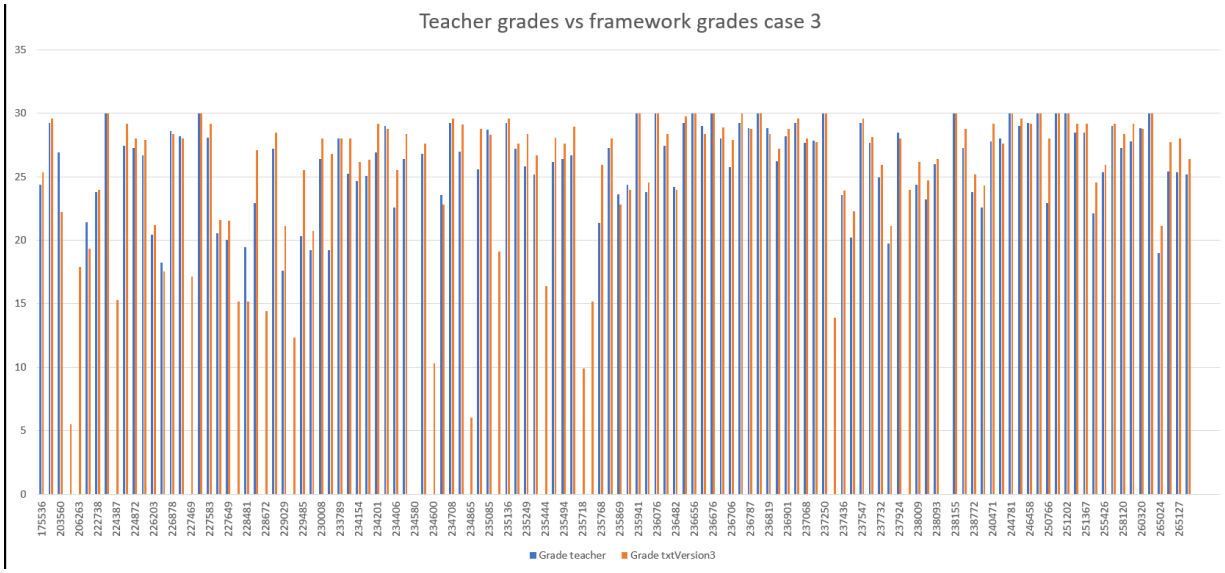

Figura 5.6: Changes method declaration, case 3

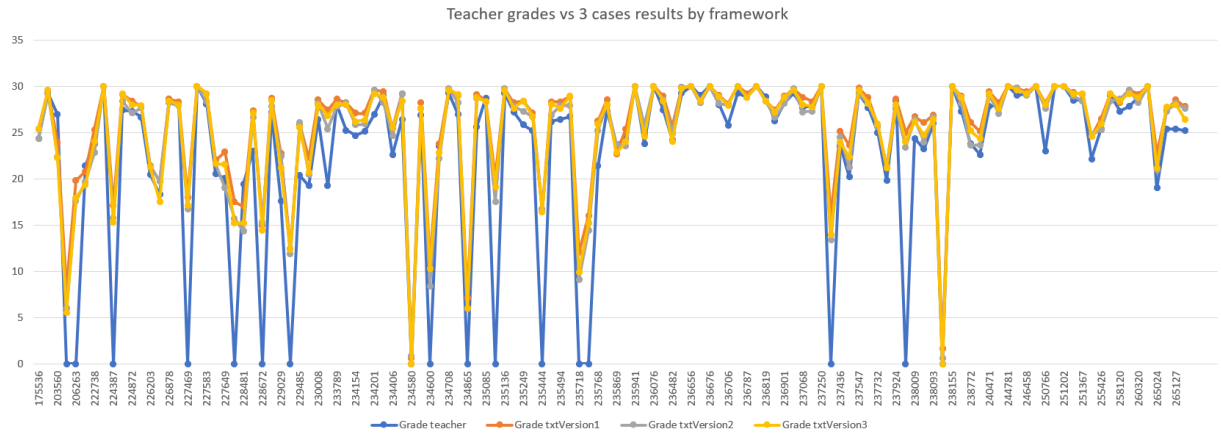

Figura 5.7: All cases

The figure 5.9 show the line graph between the grades generated by the framework in each case and the grades assigned by the teacher. Notice that the students with grade INS was included. The graphic contains only ten students for a best visualization.

The image 5.10 shows the line graph with all medias produced in the cases, with INS and without INS grades.

The image 5.11 shows the graphic with all medias generated by all grades excluded the INS grades.

5.10 5.11 5.12

For a better visualization of the results the following image 5.12 shows the comparison of the grades between teacher's grades and the grades of the framework. Also we can see that the results are close enough between them.

In the next table 5.13 shows the resume of the number of students that have pass and lost the exam according to the teacher's grade and framework's grade

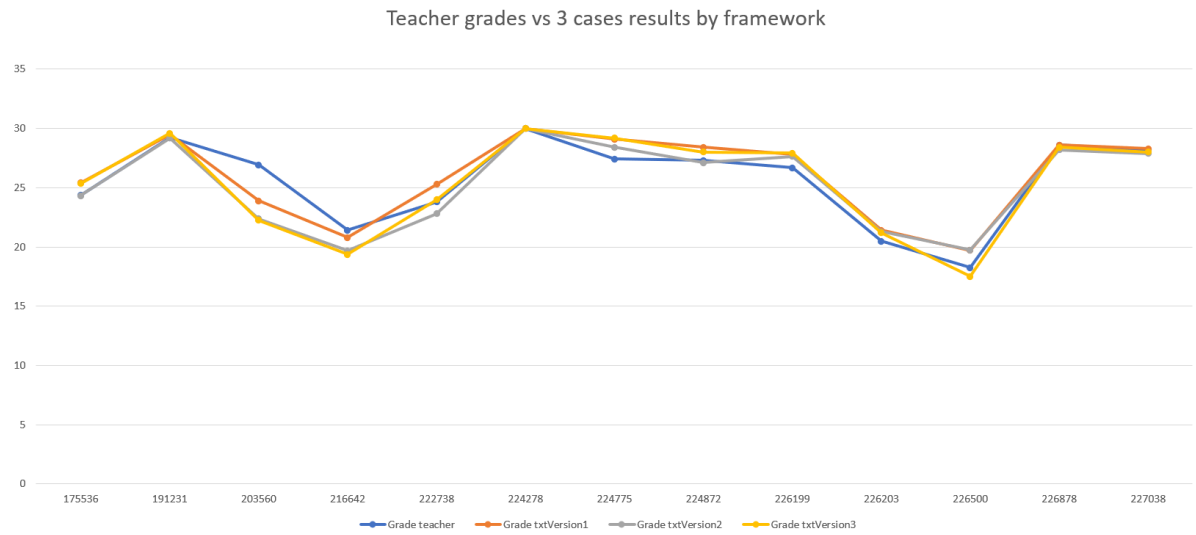

Figura 5.8: All cases without INS

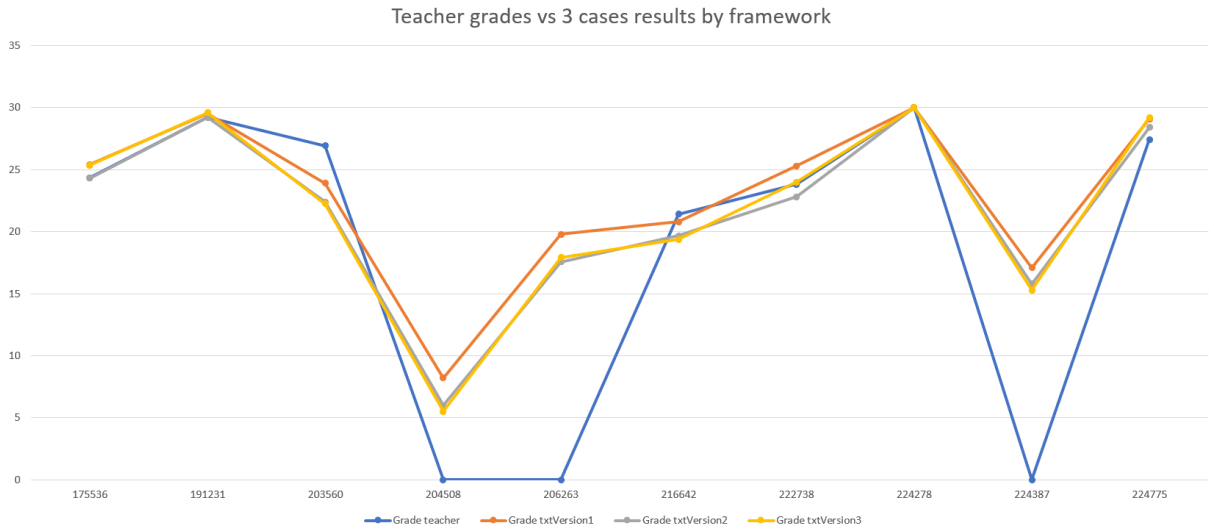

Figura 5.9: Short all cases

The successive image 5.14 shows a pie diagram of teacher's grade and the next one 5.15 shows a pie diagram of framework's grades.

the ensuring image 5.16 exhibit all the student that have passed the exam, taking on count that are kicked out the students that are not participating with the correction and all the students that are failed the exam. One more time, makes it clear that are closely to each other between teacher's grade and framework's grade.

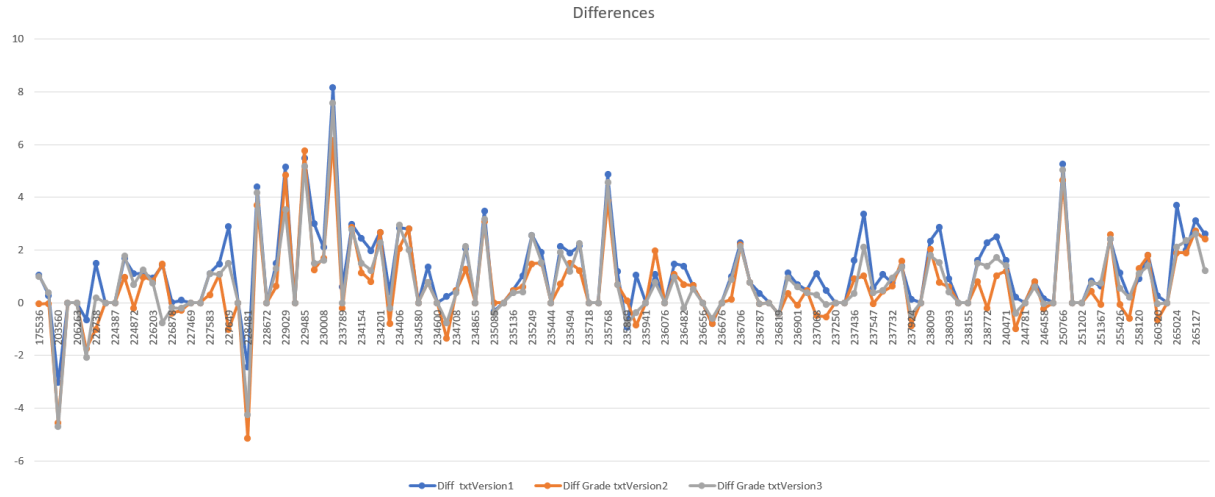

Figura 5.10: All differences between 3 cases

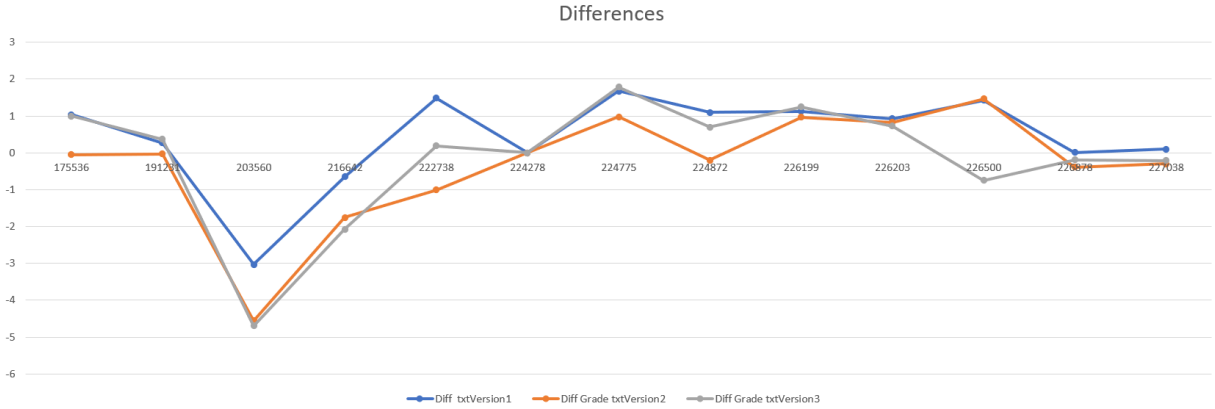

Figura 5.11: All differences without INS between 3 cases

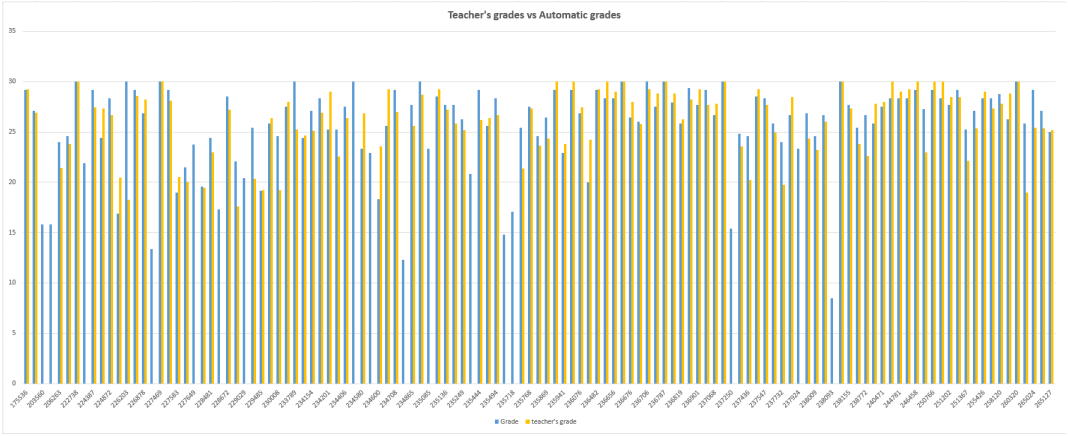

Figura 5.12: Overall grades comparison

ID STUDENT	GRADE	TEACHER'S GRADE	DIFFERENCE
175536	24,36	25,4	1,04
191231	29,23	29,5	0,27
203560	26,93	23,9	-3,02999999999999
204508	INS	8,22000000000001	0
206263	INS	19,8	0
216642	21,43	20,78	-0,649999999999988
222738	23,81	25,3	1,49
224278	30	30	0
224387	INS	17,1	0
224775	27,42	29,1	1,68
224872	27,3	28,4	1,1
226199	26,67	27,8	1,12999999999999
226203	20,47	21,4	0,930000000000014
226500	18,27	19,7	1,43000000000001
226878	28,59	28,6	0,0100000000000051
227038	28,2	28,3	0,100000000000005
227469	INS	18	0
227538	30	30	0
227583	28,09	29,2	1,11
227645	20,53	22	1,47
227649	20,02	22,9	2,88000000000001
228012	INS	17,5	0
228481	19,44	17	-2,43999999999999
228586	22,95	27,34	4,39
228672	INS	15	0
228972	27,19	28,7	1,51
229029	17,59	22,74	5,15000000000001
229159	INS	12,44	0
229485	20,33	25,82	5,49
229755	19,23	22,24	3,01000000000001
230008	26,4	28,5	2,1
230894	19,23	27,4	8,17000000000001
233789	28	28,6	0,600000000000005
233799	25,23	28,2	2,97
234154	24,66	27,1	2,44
234164	25,1	27,08	1,98
234201	26,93	29,6	2,67
234317	29	29,4	0,399999999999999
234406	22,58	25,42	2,84
234541	26,4	29,2	2,8
234580	INS	0,600000000000014	0
234587	26,83	28,2	1,37
234600	INS	11,3	0
234699	23,56	23,8	0,240000000000013
234708	29,23	29,7	0,469999999999999

Tabella 5.1: Grades professor vs grades framework case 1

ID STUDENT	GRADE	TEACHER'S GRADE	DIFFERENCE
234834	26,97	29,02	2,05
234865	INS	7,10000000000001	0
235059	25,62	29,1	3,48
235085	28,71	28,38	-0,329999999999998
235110	INS	20,2	0
235136	29,23	29,7	0,469999999999999
235163	27,19	28,2	1,01
235249	25,83	28,4	2,57
235375	25,19	27,1	1,91
235444	INS	16,6	0
235478	26,17	28,32	2,15
235494	26,4	28,3	1,9
235673	26,7	28,86	2,16
235718	INS	12	0
235726	INS	16	0
235768	21,36	26,22	4,86
235823	27,3	28,5	1,2
235869	23,62	22,7	-0,920000000000002
235887	24,36	25,4	1,04000000000001
235941	30	30	0
236027	23,8	24,88	1,08000000000001
236076	30	30	0
236447	27,42	28,9	1,48
236482	24,21	25,6	1,39
236651	29,23	29,88	0,649999999999999
236656	30	30	0
236671	29	28,2	-0,799999999999994
236676	30	30	0
236685	28	29	1
236706	25,77	28,06	2,29
236766	29,23	30	0,77
236787	28,84	29,2	0,360000000000003
236788	30	30	0
236819	28,84	28,4	-0,439999999999994
236843	26,26	27,4	1,14
236901	28,2	28,9	0,699999999999999
236987	29,23	29,7	0,469999999999999
237068	27,69	28,8	1,11
237087	27,83	28,3	0,470000000000006
237250	30	30	0
237263	INS	16,1	0
237436	23,56	25,16	1,60000000000001

Tabella 5.2: Grades professor vs grades framework case 1

ID STUDENT	GRADE	TEACHER'S GRADE	DIFFERENCE
237460	20,2	23,58	3,38000000000002
237547	29,23	29,8	0,57
237619	27,69	28,76	1,07
237732	24,97	25,74	0,77
237812	19,77	21,16	1,39
237924	28,47	28,6	0,130000000000006
237952	INS	25	0
238009	24,36	26,7	2,34000000000001
238072	23,2	26,06	2,86
238093	26	26,9	0,900000000000009
238101	INS	1,60000000000001	0
238155	30	30	0
238336	27,3	28,9	1,6
238772	23,81	26,1	2,29000000000001
239921	22,61	25,12	2,51000000000001
240471	27,8	29,4	1,6
242340	28	28,2	0,200000000000006
244781	30	30	0
246255	29	29,8	0,800000000000001
246458	29,23	29,4	0,169999999999998
246489	30	30	0
250766	22,95	28,2	5,25
251154	30	30	0
251202	30	30	0
251221	28,47	29,3	0,830000000000002
251367	28,47	29,1	0,630000000000003
253522	22,14	24,56	2,42000000000001
255426	25,37	26,5	1,13
258118	29	29,2	0,200000000000003
258120	27,3	28,2	0,900000000000006
258985	27,8	29,6	1,8
260320	28,84	29,1	0,260000000000002
265011	30	30	0
265024	19	22,7	3,70000000000001
265077	25,4	27,4	2,00000000000001
265127	25,38	28,5	3,12
266052	25,19	27,8	2,61

Tabella 5.3: Grades professor vs grades framework case 1

	teacher's grades
not pass exam	18
pass exam	106
	framework's grades
not pass exam	10
pass exam	111
not pressent correction exam	3

Figura 5.13: Data grades table

ID STUDENT	GRADE	TEACHER'S GRADE	DIFFERENCE
216642	21,43	19,6799999999999	-1,7500000000001
222738	23,81	22,7999999999999	-1,0100000000001
224278	30	30	0
224387	INS	15,7999999999999	0
224775	27,42	28,4	0,979999999999997
224872	27,3	27,0999999999999	-0,200000000000102
226199	26,67	27,6399999999999	0,969999999999899
226203	20,47	21,2999999999999	0,829999999999902
226500	18,27	19,7399999999999	1,4699999999999
226878	28,59	28,2	-0,390000000000001
227038	28,2	27,9	-0,300000000000001
227469	INS	16,7399999999999	0
227538	30	30	0
227583	28,09	28,4	0,309999999999999
227645	20,53	21,5999999999999	1,0699999999999
227649	20,02	18,9999999999999	-1,0200000000001
228012	INS	15,6999999999999	0
228481	19,44	14,2999999999999	-5,1400000000001
228586	22,95	26,6399999999999	3,6899999999999
228672	INS	15,2999999999999	0
228972	27,19	27,8199999999999	0,6299999999999
229029	17,59	22,4399999999999	4,8499999999999
229159	INS	11,8599999999999	0
229485	20,33	26,0999999999999	5,7699999999999
229755	19,23	20,4799999999999	1,2499999999999
230008	26,4	28,0999999999999	1,6999999999999
230894	19,23	25,4	6,17
233789	28	27,8	-0,199999999999999
233799	25,23	28,1	2,87
234154	24,66	25,7999999999999	1,1399999999999
234164	25,1	25,8999999999999	0,799999999999898
234201	26,93	29,6	2,67
234317	29	28,2	-0,800000000000001
234406	22,58	24,6399999999999	2,0599999999999
234541	26,4	29,2	2,8
234580	INS	0,859999999999979	0
234587	26,83	27,5999999999999	0,7699999999999
234600	INS	8,39999999999998	0
234699	23,56	22,1999999999999	-1,3600000000001
234708	29,23	29,7	0,469999999999999

Tabella 5.4: Grades professor vs grades framework case 2

ID STUDENT	GRADE	TEACHER'S GRADE	DIFFERENCE
234834	26,97	28,24	1,27
234865	INS	5,89999999999998	0
235059	25,62	28,7	3,08
235085	28,71	28,3199999999999	0
235110	INS	17,4999999999999	0
235136	29,23	29,7	0,469999999999999
235163	27,19	27,8	0,609999999999999
235249	25,83	27,2999999999999	1,4699999999999
235375	25,19	26,6999999999999	1,5099999999999
235444	INS	16,6999999999999	0
235478	26,17	26,8799999999999	0,709999999999898
235494	26,4	27,9	1,5
235673	26,7	27,9199999999999	1,2199999999999
235718	INS	9,09999999999998	0
235726	INS	14,3999999999999	0
235768	21,36	25,2399999999999	3,8799999999999
235823	27,3	28	0,699999999999999
235869	23,62	23,6999999999999	0,0799999999998988
235887	24,36	23,5	-0,859999999999999
235941	30	30	0
236027	23,8	25,7799999999999	1,9799999999999
236076	30	30	0
236447	27,42	28,5	1,08
236482	24,21	24,9	0,689999999999998
236651	29,23	29,88	0,649999999999999
236656	30	30	0
236671	29	28,2	-0,800000000000001
236676	30	30	0
236685	28	28,1199999999999	0,119999999999902
236706	25,77	27,9599999999999	2,1899999999999
236766	29,23	30	0,77
236787	28,84	28,8	-0,0399999999999991
236788	30	30	0
236819	28,84	28,4	-0,440000000000001
236843	26,26	26,5999999999999	0,339999999999897
236901	28,2	28,0999999999999	-0,100000000000101
236987	29,23	29,7	0,469999999999999
237068	27,69	27,2	-0,490000000000002
237087	27,83	27,2999999999999	-0,530000000000097
237250	30	30	0
237263	INS	13,3599999999999	0
237436	23,56	24,4599999999999	0,899999999999903

Tabella 5.5: Grades professor vs grades framework case 2

ID STUDENT	GRADE	TEACHER'S GRADE	DIFFERENCE
237460	20,2	21,2199999999999	1,0199999999999
237547	29,23	29,2	-0,0300000000000011
237619	27,69	28,12	0,43
237732	24,97	25,5999999999999	0,6299999999999
237812	19,77	21,3399999999999	1,5699999999999
237924	28,47	27,5999999999999	-0,8700000000001
237952	INS	23,3999999999999	0
238009	24,36	26,4	2,04
238072	23,2	23,9599999999999	0,759999999999902
238093	26	26,6	0,600000000000001
238101	INS	0,639999999999979	0
238155	30	30	0
238336	27,3	28,0999999999999	0,799999999999898
238772	23,81	23,5999999999999	-0,2100000000001
239921	22,61	23,6399999999999	1,0299999999999
240471	27,8	29	1,2
242340	28	27	-1
244781	30	30	0
246255	29	29,8	0,800000000000001
246458	29,23	29	-0,23
246489	30	30	0
250766	22,95	27,5999999999999	4,6499999999999
251154	30	30	0
251202	30	30	0
251221	28,47	28,9	0,43
251367	28,47	28,4	-0,0700000000000003
253522	22,14	24,72	2,58
255426	25,37	25,2999999999999	-0,0700000000000998
258118	29	28,4	-0,600000000000001
258120	27,3	28,6	1,3
258985	27,8	29,6	1,8
260320	28,84	28,2	-0,640000000000001
265011	30	30	0
265024	19	20,8999999999999	1,8999999999999
265077	25,4	27,2999999999999	1,8999999999999
265127	25,38	28,0999999999999	2,7199999999999
266052	25,19	27,6	2,41

Tabella 5.6: Grades professor vs grades framework case 2

ID STUDENT	GRADE	TEACHER'S GRADE	DIFFERENCE
175536	24,36	25,36	1,00000000000002
191231	29,23	29,6	0,370000000000001
203560	26,93	22,24	-4,68999999999997
204508	INS	5,52000000000004	0
206263	INS	17,92	0
216642	21,43	19,36	-2,06999999999996
222738	23,81	24	0,190000000000023
224278	30	30	0
224387	INS	15,28	0
224775	27,42	29,2	1,78
224872	27,3	28	0,700000000000006
226199	26,67	27,92	1,25000000000001
226203	20,47	21,2	0,730000000000032
226500	18,27	17,52	-0,74999999999995
226878	28,59	28,4	-0,189999999999994
227038	28,2	28	-0,199999999999992
227469	INS	17,1200000000001	0
227538	30	30	0
227583	28,09	29,2	1,11
227645	20,53	21,6	1,07000000000003
227649	20,02	21,52	1,50000000000004
228012	INS	15,2	0
228481	19,44	15,2	-4,23999999999995
228586	22,95	27,12	4,17000000000002
228672	INS	14,4	0
228972	27,19	28,48	1,29000000000001
229029	17,59	21,12	3,53000000000004
229159	INS	12,32	0
229485	20,33	25,52	5,19000000000002
229755	19,23	20,72	1,49000000000004
230008	26,4	28	1,60000000000001
230894	19,23	26,8	7,57000000000001
233789	28	28	0
233799	25,23	28	2,77000000000001
234154	24,66	26,16	1,50000000000001
234164	25,1	26,32	1,22000000000002
234201	26,93	29,2	2,27
234317	29	28,8	-0,199999999999996
234406	22,58	25,52	2,94000000000002
234541	26,4	28,4	2,00000000000001
234580	INS	0	0
234587	26,83	27,6	0,77000000000001
234600	INS	10,32	0
234699	23,56	22,8	-0,759999999999973
234708	29,23	29,6	0,370000000000001

Tabella 5.7: Grades professor vs Grades Framework case 3

ID STUDENT	GRADE	TEACHER'S GRADE	DIFFERENCE
234834	26,97	29,12	2,15000000000001
234865	INS	6,00000000000004	0
235059	25,62	28,8	3,18
235085	28,71	28,32	-0,38999999999999
235110	INS	19,12	0
235136	29,23	29,6	0,370000000000001
235163	27,19	27,6	0,410000000000007
235249	25,83	28,4	2,57000000000001
235375	25,19	26,72	1,53000000000002
235444	INS	16,4	0
235478	26,17	28,08	1,91000000000001
235494	26,4	27,6	1,20000000000001
235673	26,7	28,96	2,26000000000001
235718	INS	9,92000000000004	0
235726	INS	15,2	0
235768	21,36	25,92	4,56000000000002
235823	27,3	28	0,700000000000006
235869	23,62	22,8	-0,819999999999975
235887	24,36	24	-0,359999999999978
235941	30	30	0
236027	23,8	24,56	0,760000000000019
236076	30	30	0
236447	27,42	28,4	0,980000000000004
236482	24,21	24	-0,20999999999998
236651	29,23	29,76	0,530000000000001
236656	30	30	0
236671	29	28,4	-0,599999999999994
236676	30	30	0
236685	28	28,88	0,880000000000006
236706	25,77	27,92	2,15000000000001
236766	29,23	30	0,77
236787	28,84	28,8	-0,0399999999999956
236788	30	30	0
236819	28,84	28,4	-0,439999999999994
236843	26,26	27,2	0,940000000000008
236901	28,2	28,8	0,600000000000005
236987	29,23	29,6	0,370000000000001
237068	27,69	28	0,310000000000006
237087	27,83	27,76	-0,0699999999999896
237250	30	30	0
237263	INS	13,92	0
237436	23,56	23,92	0,360000000000028

Tabella 5.8: Grades professor vs Grades Framework case 3

ID STUDENT	GRADE	TEACHER'S GRADE	DIFFERENCE
237460	20,2	22,32	2,12000000000003
237547	29,23	29,6	0,370000000000001
237619	27,69	28,16	0,470000000000006
237732	24,97	25,92	0,950000000000021
237812	19,77	21,12	1,35000000000004
237924	28,47	28	-0,469999999999992
237952	INS	24	0
238009	24,36	26,16	1,80000000000001
238072	23,2	24,72	1,52000000000002
238093	26	26,4	0,400000000000013
238101	INS	0	0
238155	30	30	0
238336	27,3	28,8	1,5
238772	23,81	25,2	1,39000000000002
239921	22,61	24,32	1,71000000000002
240471	27,8	29,2	1,4
242340	28	27,6	-0,399999999999991
244781	30	30	0
246255	29	29,6	0,600000000000001
246458	29,23	29,2	-0,0299999999999976
246489	30	30	0
250766	22,95	28	5,05000000000001
251154	30	30	0
251202	30	30	0
251221	28,47	29,2	0,730000000000004
251367	28,47	29,2	0,730000000000004
253522	22,14	24,56	2,42000000000002
255426	25,37	25,92	0,550000000000018
258118	29	29,2	0,200000000000003
258120	27,3	28,4	1,1
258985	27,8	29,2	1,4
260320	28,84	28,8	-0,0399999999999956
265011	30	30	0
265024	19	21,12	2,12000000000004
265077	25,4	27,76	2,36000000000001
265127	25,38	28	2,62000000000001
266052	25,19	26,4	1,21000000000001

Tabella 5.9: Grades professor vs Grades Framework case 3

ID STUDENT	GRADE	TEACHER'S GRADE
191231	22,2975	24,36
203560	29,1675	29,23
204508	27,085	26,93
206263	15,8375	0
216642	15,84	0
222738	23,9625	21,43
224278	24,5875	23,81
224387	30	30
224775	21,88	0
224872	29,1675	27,42
226199	24,3775	27,3
226203	28,335	26,67
226500	16,885	20,47
226878	30	18,27
227038	29,1675	28,59
227469	26,8775	28,2
227538	13,345	0
227583	30	30
227645	29,1675	28,09
227649	18,965	20,53
228012	21,4625	20,02
228481	23,755	0
228586	19,5875	19,44
228672	24,3775	22,95
228972	17,3	0
229029	28,5425	27,19
229159	22,085	17,59
229485	20,42	0
229755	25,42	20,33
230008	19,175	19,23
230894	25,835	26,4
233799	27,5025	28
234154	30	25,23
234164	24,38	24,66
234201	27,085	25,1
234317	28,335	26,93
234406	25,2125	29
234541	25,21	22,58
234580	27,5025	26,4
234587	30	0
234600	23,335	26,83
234699	22,92	0
234708	18,3425	23,56
234834	25,625	29,23
234865	29,1675	26,97

Tabella 5.10: Grades professor vs Grades Framework

ID STUDENT	GRADE	TEACHER'S GRADE
235059	12,2975	0
235085	27,71	25,62
235110	30	28,71
235136	23,335	0
235163	28,5425	29,23
235249	27,71	27,19
235375	27,71	25,83
235444	26,2525	25,19
235478	20,8375	0
235494	29,1675	26,17
235673	25,6275	26,4
235718	28,335	26,7
235726	14,7975	0
235768	17,09	0
235823	25,42	21,36
235869	27,5	27,3
235887	24,585	23,62
235941	26,46	24,36
236027	29,1675	30
236076	22,9225	23,8
236447	29,1675	30
236482	26,8775	27,42
236651	20,0075	24,21
236656	29,1675	29,23
236671	28,335	30
236676	28,335	29
236685	30	30
236706	26,46	28
236766	26,045	25,77
236787	30	29,23
236788	27,5025	28,84
236819	30	30
236843	27,9175	28,84
236901	25,835	26,26
236987	29,375	28,2
237068	27,71	29,23
237087	29,1675	27,69
237250	26,67	27,83
237263	30	30
237436	15,425	0
237460	24,795	23,56

Tabella 5.11: Grades professor vs Grades Framework

ID STUDENT	GRADE	TEACHER'S GRADE
237547	24,5875	20,2
237619	28,5425	29,23
237732	28,335	27,69
237812	25,8375	24,97
237924	23,9625	19,77
237952	26,67	28,47
238009	23,335	0
238072	26,8775	24,36
238093	24,585	23,2
238101	26,67	26
238155	8,45000000000006	0
238336	30	30
238772	27,71	27,3
239921	25,42	23,81
240471	26,6675	22,61
242340	25,8375	27,8
244781	27,5025	28
246255	28,335	30
246458	28,335	29
246489	28,335	29,23
250766	29,1675	30
251154	27,2925	22,95
251202	29,1675	30
251221	28,335	30
251367	27,71	28,47
253522	29,1675	28,47
255426	25,2125	22,14
258118	27,085	25,37
258120	28,335	29
258985	28,335	27,3
260320	28,75	27,8
265011	26,2525	28,84
265024	30	30
265077	25,835	19
265127	29,1675	25,4
266052	27,085	25,38
	25,0025	25,19

Tabella 5.12: Grades professor vs Grades Framework

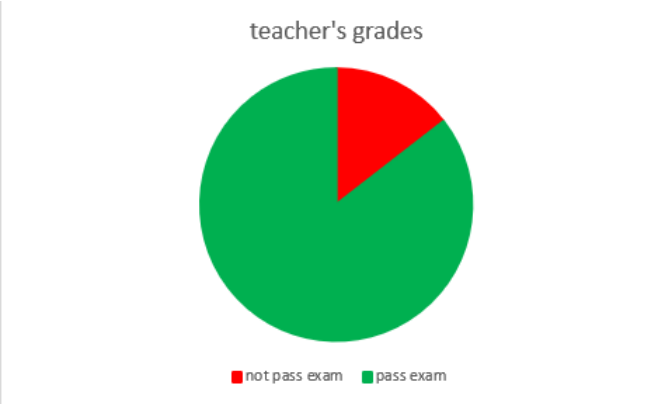

Figura 5.14: Teacher grades

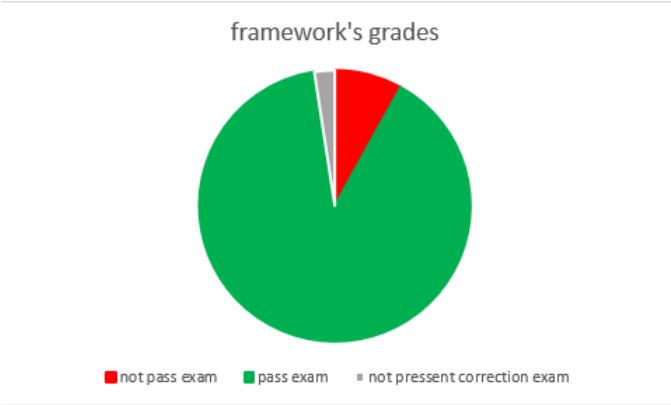

Figura 5.15: Framework grades

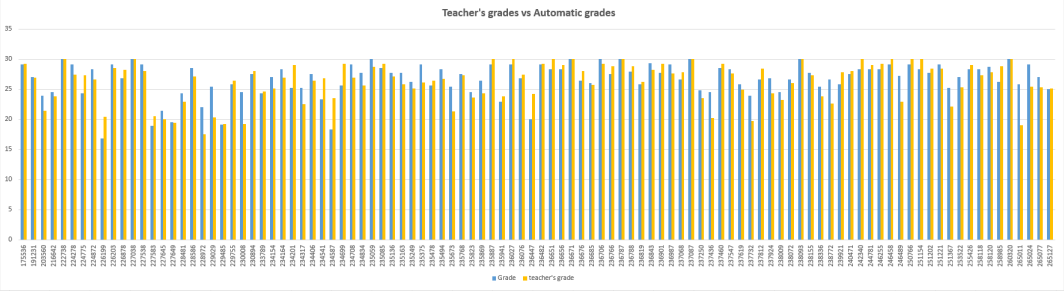

Figura 5.16: Passed Exam

Capitolo 6

Conclusion

The main goal of this thesis is the development of a framework that helps the teacher with the revision process of two javas projects. The assignment of the grades to the student is based on the difference presented between the projects, this objective is reached, presented to the good accuracy in comparison by the grades assignment by the teachers and in comparison with other framework that are used in the course for the activity of evaluation.

The procedure of the system , the students delivered the first project and a second version that is an incentive for correct to error in the first delivery, so with this technique the student improves the solution and learns about his error.

For the professor is a tool that automates the evaluation process and saves time, because in the course of java object-oriented programming, the number of students that take this course in the bachelor degree is considerably high so the time for checking the work in the projects of all students is very tricky. Another important feature that this framework gets to the teachers, is the possibility of discrimination by specific parts of the code or specific nodes and assignment to this node a grade major.

Other people that get a profit from the implementation of this framework are the students, after that, they deliver the two versions of the project. The students can get feedback about the mistakes found and how the grade was assigned so that students can learn about the mistakes and how they influence the final grade.

6.1 Future works

In future works, the mechanism for navigating the abstract syntax trees as much in one treeA as the other tree must be improved, because the navigate in the AST take a little amount of time to be done all comparisons but maybe if the number students is increasing, it is possible that the time too increasing.

Nowadays a important task in the area of software is the work in a team, and the update of the source code is a crucial step where the programmers have many headaches.This framework give the opportunity to reuse the code in order to manages the differences in a correct way in merging time.

Bibliografia

[1] *Eclipse official website*. URL: http://eclipse.org/.

[2] *Eclipse official website*. URL: https://www.eclipse.org/articles/article.php?file=Article-JavaCodeManipulation_AST/index.html.

[3] *GumTree*. URL: https://dl.acm.org/doi/10.1145/2642937.2642982.

[4] *GumTreeAlgorithm*. URL: https://courses.cs.vt.edu/cs6704/spring17/slides_by_students/CS6704_gumtree_Kijin_AN_Feb15.pdf.

[5] *IEEE article ClDiff: Generating Concise Linked Code Differences*. URL: https://ieeexplore.ieee.org/abstract/document/9000085.

[6] *Java AST IBM website*. URL: https://www.ibm.com/docs/en/rsar/9.5?topic=SS5JSH_9.5.0/org.eclipse.jdt.doc.isv/reference/api/org/eclipse/jdt/core/dom/AST.htm.

[7] *Java official website*. URL: http://www.java.com/it/.

[8] *JUnit official website*. URL: https://junit.org/junit5/.

[9] Giorgio Bruno Marco Torchiano. "Integrating Software Engineering Key Practices into an OOP Massive In-Classroom Course: an Experience Report". In: (2018), p. 9.

[10] IBM Ottawa Lab Thomas Kuhn Eye Media GmbH Olivier Thomann. "JavaCodeManipulation$_{A}$ In: (2006).

[11] IBM Ottawa Lab Thomas Kuhn Eye Media GmbH Olivier Thomann. "Version control with subversion". In: *O'Reilly Media, Inc.* (2004).

[12] *w3schools*. URL: https://www.w3schools.com/java/java_oop.asp.