Matching PMD Alerts

Bruno Crotman

04/05/2020

Contents

1 Introduction

This document is part of a larger research project about software degradation caused by careless developers' behavior and strategies to deal with such undesired behavior. The strategies to deal with this problem will possibly be inspired by concepts from game theory. At this moment, we assume that software degradation can be measured by the number and the types of kludges made by software developers in the code. So, one of the goals of this project is to study how software projects evolve in terms of number and kinds of kludges. Right now, we are trying to identify kludges by looking at alerts generated by the PMD source code analyzer.

To evaluate how the number of alerts evolves throughout the history of a software project, we must be able to analyze two different versions of a source code module (an old and a new version) and categorize each alert contained in the new version as either **new**, **fixed** or **open**.

A PMD alert generated for the old version is either **open** or **fixed** in the new version. An **open** alert remains in the new version of the code. A **fixed** alert does not exist in the new version.

A PMD alert generated for the new version is either **open** or **new**. An **open** alert indicates that the same alert was identified in the old version. A **new** alert implies that the same alert cannot be identified in the old version.

The alerts identified as **open** are equivalent in both new and old versions. To decide whether an alert is **open**, **fixed** or **new**, one has to identify if an alert in the old version is equivalent to an alert in the new version. The intersection between **fixed** alerts, **new** alerts and **open** alerts is empty.

In order to decide if an alert is **open**, **fixed** or **new**, we have to identify if an alert in the old version is equivalent to an alert in the new version. This document describes the algorithms we are using to make this

classification.

In Section 2, I describe how I use PMD source code analyzer in two tasks. The first task is to list the alerts that represent possible kludges. PMD receives a source code (Prof Márcio, ele pode gerar alertas de um software inteiro, não só um conjunto, por isso tirei "module") as input and generates a list of bad programming practices contained in the code. The process we follow to generate the alerts using PMD source code analyzer is discussed in Section 2.1. The other task for which we use PMD is in the creation of an Abstract Syntax Tree (AST) from a source code with selected nodes. This will help us in one of the algorithms described in Section 3. The creation of the AST using PMD is described in Section 2.2.

In Section 3, I describe two algorithms that categorize the alerts as **new**, **fixed** or **open**. The first one, described in Section 3.1, is a naive algorithm based on matches by lines of code and some key features of the alerts. The second is a more sophisticated algorithm, based on matches by blocks of code, using the Abstract Syntax Tree.

2 PMD Source Code Analyzer

PMD is static source code analyzer that is commonly used to find possible programming flaws. In this work we use PMD for two tasks: generate alerts that we interpret as clues about kludges and create an AST from the source code with selected kinds of node.

2.1 Using PMD to generate alerts

PMD traverses the AST of a source code searching for violations of rules that are configured by the user. PMD comes with a default rule set for Java language. The default rule set finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth. It´s possible configure a different set of rules creating a custom XML file. At this point we use the default rule set to generate the alerts that we interpret as kludges and try to categorize in Section 3

In Figure 1, we can see an example of a simple code and the alerts that were generated by the default rule set of PMD alerts tool.

```
/*  1-                                  */package pack_x;
/*  2-                                  */
/*  3-                                  */import importX.function;
/*  4-                                  */
/*  5-                                  */class ClassX extends ClassY implements InterfX {
/*  6-                                  */    private long fieldX;
/*  7-                                  */
/*  8-                                  */    ClassX(int paramX, double paramY) {
/*  9-                                  */        int varX = function(paramX, paramY);
/* 10-                                  */        if (varX == 0)
/* 11-ControlStatementBraces            */            this.fieldX = 1;
/* 12-                                  */        else{
/* 13-                                  */            this.fieldX = 0;
/* 14-                                  */     }
/* 15-                                  */    }
/* 16-                                  */    @Override
/* 17-                                  */    public int methodX(int paramW, Boolean paramZ)
/* 18-                                  */    {
/* 19-                                  */        if (paramZ)
/* 20-ControlStatementBraces            */            fieldX = paramW;
/* 21-                                  */        else{
/* 22-                                  */            fieldX = 0;
/* 23-                                  */     }
/* 24-                                  */        return paramW + this.fieldX;
/* 25-                                  */     }
/* 26-                                  */}
```

Figure 1: Simple code with its alerts

2.2 Using PMD to generate an Abstract Syntax Tree

The AST is used, in conjunction with the relation between the lines we will see in Section 3.1.3, to understand the location of an alert in a version of a code. We use this information in the algorithm described in Section 3.2

PMD lets us configure our own rules. Figure 2 shows the designer tool that helps a user to create custom rules.

We can see that the tool traverses the source code visiting many different kinds of elements. If we build our own simple rules, aimed only to capture some kinds of elements, we will generate list of “alerts” that will contain all the elements of the chosen kinds.

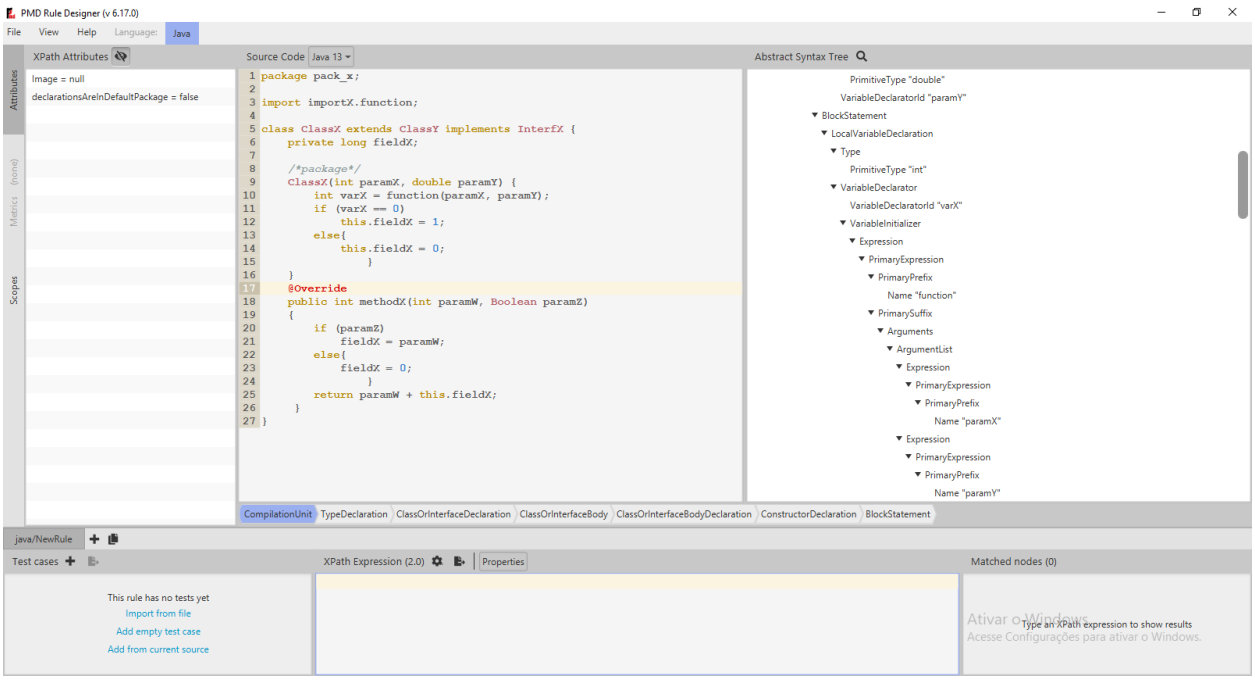

Figure 2: PMD Designer tool

In Figure 3, we show an example of a ruleset that captures all the method declarations.

```
<?xml version="1.0"?>
<ruleset name="complete"
         xmlns="http://pmd.sourceforge.net/ruleset/2.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://pmd.sourceforge.net/ruleset/2.0.0 https://pmd.sourceforge.io/ruleset_2_0_0.xsd"
    <description>Test.</description>

    <rule name="method"
      language="java"
      message="method"
      class="net.sourceforge.pmd.lang.rule.XPathRule" >
    <description>
TODO
    </description>
    <priority>3</priority>
    <properties>
        <property name="xpath">
            <value>
<![CDATA[
//MethodDeclaration
]]>
            </value>
        </property>
    </properties>
    </rule>
]

</ruleset>
```

Figure 3: Custom ruleset for PMD alerts tool

Table 1 shows the kinds of elements that were selected for the creation of an AST for the code in Figure 1.

Table 1: Kinds of elements selected

rule_number	rule
1	annotation
2	block
3	class_or_interface_body
4	class_or_interface_declaration
5	class_or_interface_type
6	compilation_unit
7	constructor_declaration
8	extends_list
9	field_declaration
10	formal_parameter
11	formal_parameters
12	if_statement
13	implements_list
14	import_declaration
15	method
16	name
17	package
18	statement
19	type_declaration
20	variable_id

If we select the list in Table 1, the simple code shown in this Section captures the elements shown in Table 2

Table 1 contains the list and location of the elements of the AST. In order to recreate the AST, we must follow three steps:

1. Link each element a to the set of elements X that are fully located between the begin line / begin column and end line / end column of element a. We can construct a directed graph in which the elements are the nodes and the links are the edges. This is not a tree yet, because each node will have edges directed to all its descendants and not only its children in the AST.
2. Sort the nodes in the decreasing order of its number of children. The objective is to establish that, in a search through this graph, the first child chosen will be the one that is a child in the AST, and not only on this graph.
3. Proceed a deep-first search starting from the compilation unit node.

Table 2: Elements captured in code

line	endline	col	endcol	rule	method	code
1	26	1	3	compilation_unit	No method	
1	1	1	15	package	No method	
1	1	9	14	name	No method	
3	3	1	24	import_declaration	No method	
3	3	8	23	name	No method	
5	26	1	1	class_or_interface_declaration	No method	
5	5	14	27	extends_list	No method	
5	5	22	27	class_or_interface_type	No method	
5	5	29	46	implements_list	No method	
5	5	40	46	class_or_interface_type	No method	
5	26	48	1	class_or_interface_body	No method	
6	6	13	24	field_declaration	No method	
6	6	18	23	variable_id	No method	
8	15	5	5	constructor_declaration	ClassX	
8	8	11	37	formal_parameters	ClassX	
8	8	12	21	formal_parameter	ClassX	
8	8	16	21	variable_id	ClassX	
8	8	24	36	formal_parameter	ClassX	
8	8	31	36	variable_id	ClassX	
9	9	13	16	variable_id	ClassX	
9	9	20	27	name	ClassX	
9	9	29	34	name	ClassX	
9	9	37	42	name	ClassX	
10	14	9	17	statement	ClassX	
10	14	9	17	if_statement	ClassX	
10	10	13	16	name	ClassX	
11	11	13	28	statement	ClassX	
12	14	13	17	block	ClassX	
12	14	13	17	statement	ClassX	
13	13	13	28	statement	ClassX	
16	16	5	13	annotation	No method	
16	16	6	13	name	No method	
17	25	12	6	method	methodX	
17	17	23	50	formal_parameters	methodX	
17	17	24	33	formal_parameter	methodX	
17	17	28	33	variable_id	methodX	
17	17	36	42	class_or_interface_type	methodX	
17	17	36	49	formal_parameter	methodX	
17	17	44	49	variable_id	methodX	
18	25	5	6	block	methodX	
19	23	9	17	statement	methodX	
19	23	9	17	if_statement	methodX	
19	19	13	18	name	methodX	
20	20	13	28	statement	methodX	
20	20	13	18	name	methodX	
20	20	22	27	name	methodX	
21	23	13	17	block	methodX	
21	23	13	17	statement	methodX	
22	22	13	23	statement	methodX	
22	22	13	18	name	methodX	
24	24	9	36	statement	methodX	
24	24	16	21	name	methodX	

After we follow these steps, we come up with the AST as we see in Figure 4.

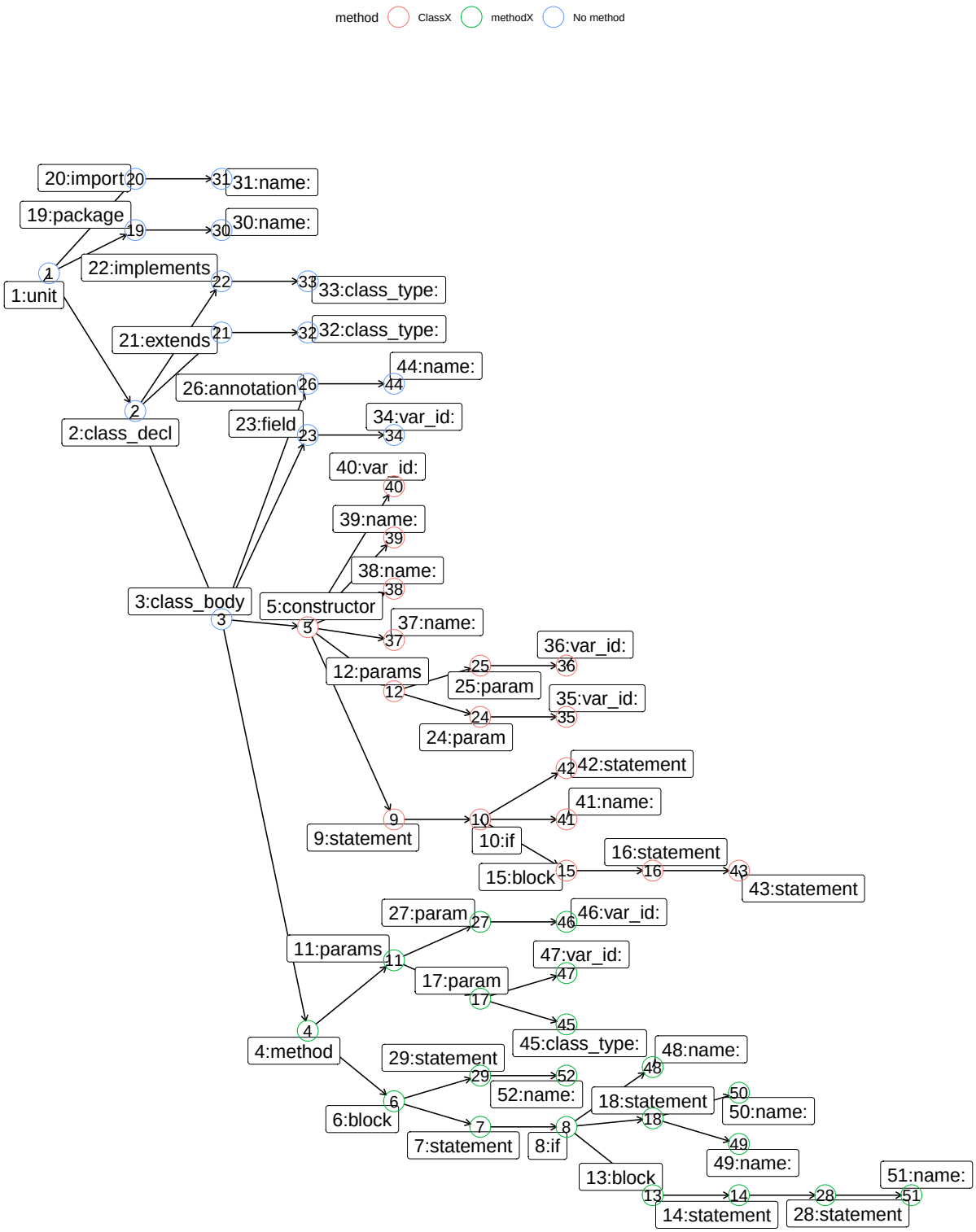

Figure 4: Abstract Syntax Tree

3 Algorithms to categorize alerts

3.1 Matches by line of code

In this first algorithm, I match the lines of code of the old version with the lines of code of the new version using information from the output of git's diff command. When an alert with the same features occurs in both matched lines, this alert is declared **open**. The alerts that occur in a not matched line of the old version are declared *fixed* and the alerts located in a unmatched line of the new version are declared as **new**.

These are the steps of the algorithm:

1. Generate a list of alerts from each version (old and new) using PMD Alert (Section 3.1.1);
2. Generate the git diff between the two versions (Section 3.1.2);
3. Using information from git diff, create a map between the lines (Section 3.1.3);
4. Categorize the alerts (Section 3.1.4).

3.1.1 Generate a list of alerts for each version

The two codes presented in this Section, named "new version" and "old version", are used in Sections 3.1.2, 3.1.3 and 3.1.4 to describe the algorithm.

The old version, with the alerts generated by PMI, is shown in Figure 5.

```
/*  1-                                  */package twitter4j;
/*  2-                                  */
/*  3-UnusedImports                     */import java.util.concurrent.ConcurrentHashMap;
/*  4-                                  */
/*  5-                                  */class TwitterImpl extends TwitterBaseImpl implements Twitter {
/*  6-                                  */    private static final long serialVersionUID = 91709430840960857 70L;
/*  7-UnusedPrivateField                */    private static final Logger logger = Logger.getLogger(TwitterBaseImpl.class);
/*  8-                                  */
/*  9-                                  */    TwitterImpl(Configuration conf, Authorization auth) {
/* 10-                                  */        super(conf, auth);
/* 11-                                  *//* ...  */
/* 44-                                  *//* ...  */
/* 45-                                  */            if (conf.isTweetModeExtended()) {
/* 46-                                  */                params.add(new HttpParameter("tweet_mode", "extended"));
/* 47-                                  */            }
/* 48-OptimizableToArrayCall            */            HttpParameter[] implicitParams = params.toArray(new HttpParameter[params.size()]);
/* 49-                                  */
/* 50-                                  */            // implicitParamsMap.containsKey() is evaluated in the above if clause.
/* 51-                                  */            // thus implicitParamsStrMap needs to be initialized first
/* 52-                                  *//* ...  */
/* 57-                                  *//* ...  */
/* 58-                                  */    }
/* 59-                                  */
/* 60-                                  */    @Override
/* 61-FormalParameterNamingConventions  */    public AccountSettings updateAccountSettings(Integer trend_locationWoeid,
/* 62-FormalParameterNamingConventions(2)*/                                                 Boolean sleep_timeEnabled, String start_sleepTime,
/* 63-FormalParameterNamingConventions(2)*/                                                 String end_sleepTime, String time_zone, String lang)
/* 64-                                  */            throws TwitterException {
/* 65-                                  */        List<HttpParameter> profile = new ArrayList<HttpParameter>(6);
/* 66-                                  */        if (trend_locationWoeid != null) {
/* 67-                                  *//* ...  */
/* 81-                                  *//* ...  */
/* 82-                                  */            profile.add(new HttpParameter("lang", lang));
/* 83-                                  */        }
/* 84-                                  */        return factory.createAccountSettings(post(conf.getRestBaseURL() + "account/settings.json"
/* 85-OptimizableToArrayCall            */                , profile.toArray(new HttpParameter[profile.size()])));
/* 86-                                  */
/* 87-                                  */    }
/* 88-                                  */}
```

Figure 5: Example: old version

Table 3 lists the alerts found in the old version.

Table 3: Alerts in the old version

id	beginline	ruleset	rule	package	class	method	variable
1	3	Best Practices	UnusedImports	twitter4j	TwitterImpl	No method	No variable
2	7	Best Practices	UnusedPrivateField	twitter4j	TwitterImpl	No method	logger
3	48	Performance	OptimizableToArrayCall	twitter4j	TwitterImpl	TwitterImpl	No variable
4	61	Code Style	FormalParameterNamingConventions	twitter4j	TwitterImpl	updateAccountSettings	trend_locationWoeid
5	62	Code Style	FormalParameterNamingConventions	twitter4j	TwitterImpl	updateAccountSettings	sleep_timeEnabled
6	62	Code Style	FormalParameterNamingConventions	twitter4j	TwitterImpl	updateAccountSettings	start_sleepTime
7	63	Code Style	FormalParameterNamingConventions	twitter4j	TwitterImpl	updateAccountSettings	end_sleepTime
8	63	Code Style	FormalParameterNamingConventions	twitter4j	TwitterImpl	updateAccountSettings	time_zone
9	85	Performance	OptimizableToArrayCall	twitter4j	TwitterImpl	updateAccountSettings	No variable

The new version, shown in Figure 6, has the alerts listed in Table 4.

Table 4: Alerts in the new version

id	beginline	ruleset	rule	package	class	method	variable
1	5	Best Practices	UnusedPrivateField	twitter4j	TwitterImpl	No method	logger
2	47	Performance	OptimizableToArrayCall	twitter4j	TwitterImpl	TwitterImpl	No variable
3	62	Code Style	FormalParameterNamingConventions	twitter4j	TwitterImpl	updateAccountSettings	sleep_timeEnabled
4	62	Code Style	FormalParameterNamingConventions	twitter4j	TwitterImpl	updateAccountSettings	start_sleepTime
5	63	Code Style	FormalParameterNamingConventions	twitter4j	TwitterImpl	updateAccountSettings	end_sleepTime
6	63	Code Style	FormalParameterNamingConventions	twitter4j	TwitterImpl	updateAccountSettings	time_zone
7	85	Performance	OptimizableToArrayCall	twitter4j	TwitterImpl	updateAccountSettings	No variable
8	89	Best Practices	UnusedPrivateField	twitter4j	TwitterImpl	No method	not_used

By mapping the alerts in both version, we observe the following:

- line 3 in the old version had a “Unused Import” which was removed in the new version. So, the alert related to this line must be declared **fixed**;
- line 63 in the old version (61 in the new version) was fixed by changing the name of the parameter. This must be classified as another **fixed** alert;
- line 89 was added to the new version and contains a unused private field. Such must be categorized as a **new** alert.

```
/*  1-                                  */package twitter4j;
/*  2-                                  */
/*  3-                                  */class TwitterImpl extends TwitterBaseImpl implements Twitter {
/*  4-                                  */    private static final long serialVersionUID = 9170943084096085770L;
/*  5-UnusedPrivateField                */    private static final Logger logger = Logger.getLogger(TwitterBaseImpl.class);
/*  6-                                  */
/*  7-                                  */    /*package*/
/*  8-                                  */    TwitterImpl(Configuration conf, Authorization auth) {
/*  9-                                  *//* ...  */
/* 43-                                  *//* ...  */
/* 44-                                  */            if (conf.isTweetModeExtended()) {
/* 45-                                  */                params.add(new HttpParameter("tweet_mode", "extended"));
/* 46-                                  */            }
/* 47-OptimizableToArrayCall            */            HttpParameter[] implicitParams = params.toArray(new HttpParameter[params.size()]);
/* 48-                                  */
/* 49-                                  */            // implicitParamsMap.containsKey() is evaluated in the above if clause.
/* 50-                                  */            // thus implicitParamsStrMap needs to be initialized first
/* 51-                                  *//* ...  */
/* 58-                                  *//* ...  */
/* 59-                                  */
/* 60-                                  */    @Override
/* 61-                                  */    public AccountSettings updateAccountSettings(Integer trendlocationWoeid,
/* 62-FormalParameterNamingConventions(2)*/                                                 Boolean sleep_timeEnabled, String start_sleepTime,
/* 63-FormalParameterNamingConventions(2)*/                                                 String end_sleepTime, String time_zone, String lang)
/* 64-                                  */            throws TwitterException {
/* 65-                                  */        List<HttpParameter> profile = new ArrayList<HttpParameter>(6);
/* 66-                                  */        if (trendlocationWoeid != null) {
/* 67-                                  *//* ...  */
/* 81-                                  *//* ...  */
/* 82-                                  */            profile.add(new HttpParameter("lang", lang));
/* 83-                                  */        }
/* 84-                                  */        return factory.createAccountSettings(post(conf.getRestBaseURL() + "account/settings.json"
/* 85-OptimizableToArrayCall            */                , profile.toArray(new HttpParameter[profile.size()])));
/* 86-                                  */
/* 87-                                  */    }
/* 88-                                  */
/* 89-UnusedPrivateField                */    private int not_used = 0;
/* 90-                                  */
/* 91-                                  */}
```

Figure 6: Example: new version

Table 5: Relation between lines of the old version and lines of the new version

1	2	3	4	5	6	7		8	9	10-	-56	57	58		59	60		61	62	63	64	65		66		67	68	69	70-	-85	86	87				88
1	2			3	4	5	7	6	8	9-	-55	56	57	58	59	60	61		62	63	64	65	66		67		68	69	70-	-85	86	87	88	89	90	91

3.1.2 Generate the git diff between the two versions

The *git diff* command is executed between two subsequent versions with the option –patience. The result of the *git diff* operation between the versions presented in the last section is shown in Figure 7.

```
8   5   j/match_algorithm_description/{old => new}/code.java

diff --git a/j/match_algorithm_description/old/code.java b/j/match_algorithm_description/new/code.java
index f1a64fa..245a6e8 100644
--- a/j/match_algorithm_description/old/code.java
+++ b/j/match_algorithm_description/new/code.java
@@ -3,2 +2,0 @@ package twitter4j;
-import java.util.concurrent.ConcurrentHashMap;
-
@@ -8,0 +7 @@ class TwitterImpl extends TwitterBaseImpl implements Twitter {
+    /*package*/
@@ -58,0 +58 @@ class TwitterImpl extends TwitterBaseImpl implements Twitter {
+
@@ -61 +61 @@ class TwitterImpl extends TwitterBaseImpl implements Twitter {
-    public AccountSettings updateAccountSettings(Integer trend_locationWoeid,
+    public AccountSettings updateAccountSettings(Integer trendlocationWoeid,
@@ -66,2 +66,2 @@ class TwitterImpl extends TwitterBaseImpl implements Twitter {
-        if (trend_locationWoeid != null) {
-            profile.add(new HttpParameter("trend_location_woeid", trend_locationWoeid));
+        if (trendlocationWoeid != null) {
+            profile.add(new HttpParameter("trend_location_woeid", trendlocationWoeid));
@@ -87,0 +88,3 @@ class TwitterImpl extends TwitterBaseImpl implements Twitter {
+
+    private int not_used = 0;
+
```

Figure 7: Git diff between code in Figure 5 and code in Figure 6

3.1.3 Using information from git diff, create a relation between the lines

For each difference stated in the output (the sections of the diff file starting with "@@"), there is an indication of the number of lines removed from the old version and the number of lines added to the new one. The line in which the lines are removed from the old version and the line at which the lines are added is indicated, too. By using this information we create a relation between the lines of the old version and the equivalent lines in the new version. For the new and old versions presented in Section 3.1.1, the relation is shown in Table 5.

Now we can connect the two versions, as well as their alerts, as we see in Figure 8.

```
/*  1-                        */package twitter4j;
/*  2-                        */
/*  3-UnusedImports           */import java.util.concurrent.ConcurrentHashMap;
/*  4-                        */
/*  5-                        */class TwitterImpl extends TwitterBaseImpl implements Twitter {
/*  6-                        */    private static final long serialVersionUID = 9170943084096085770L;
/*  7-UnusedPrivateField      */    private static final Logger logger = Logger.getLogger(TwitterBaseImpl.class);
/*   -                        *//*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/*  8-                        */
/*  9-                        */    TwitterImpl(Configuration conf, Authorization auth) {
/* 10-                        *//* ...  */
/* 44-                        *//* ...  */
/* 45-                        */            if (conf.isTweetModeExtended()) {
/* 46-                        */                params.add(new HttpParameter("tweet_mode", "extended"));
/* 47-                        */            }
/* 48-OptimizableToArrayCa    */            HttpParameter[] implicitParams = params.toArray(new HttpParameter[params.size(
/* 49-                        */
/* 50-                        */            // implicitParamsMap.containsKey() is evaluated in the above if clause.
/* 51-                        */            // thus implicitParamsStrMap needs to be initialized first
/* 52-                        *//* ...  */
/*   -                        *//* ...  */
/* 59-                        */
/* 60-                        */    @Override
/*   -                        *//*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/* 61-FormalParameterNamin    */    public AccountSettings updateAccountSettings(Integer trend_locationWoeid,
/* 62-FormalParameterNamin(2)*/                                                 Boolean sleep_timeEnabled, String start_s
/* 63-FormalParameterNamin(2)*/                                                 String end_sleepTime, String time_zone, S
/* 64-                        */            throws TwitterException {
/* 65-                        */        List<HttpParameter> profile = new ArrayList<HttpParameter>(6);
/*   -                        *//*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/* 66-                        *//* ...  */
/* 81-                        *//* ...  */
/* 82-                        */            profile.add(new HttpParameter("lang", lang));
/* 83-                        */        }
/* 84-                        */        return factory.createAccountSettings(post(conf.getRestBaseURL() + "account/setting
/* 85-OptimizableToArrayCa    */                , profile.toArray(new HttpParameter[profile.size()])));
/* 86-                        */
/* 87-                        */    }
/*   -                        *//*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/*   -                        *//*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/*   -                        *//*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/* 88-                        */}
```

```
/*  1-                        */package twitter4j;
/*  2-                        */
/*   -                        *//*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/*   -                        *//*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/*  3-                        */class TwitterImpl extends TwitterBaseImpl implements Twitter {
/*  4-                        */    private static final long serialVersionUID = 9170943084096085770L;
/*  5-UnusedPrivateField      */    private static final Logger logger = Logger.getLogger(TwitterBaseImpl.class);
/*  7-                        */    /*package*/
/*  6-                        */
/*  8-                        */    TwitterImpl(Configuration conf, Authorization auth) {
/*  9-                        *//* ...  */
/* 43-                        *//* ...  */
/* 44-                        */            if (conf.isTweetModeExtended()) {
/* 45-                        */                params.add(new HttpParameter("tweet_mode", "extended"));
/* 46-                        */            }
/* 47-OptimizableToArrayCa    */            HttpParameter[] implicitParams = params.toArray(new HttpParameter[params.size(
/* 48-                        */
/* 49-                        */            // implicitParamsMap.containsKey() is evaluated in the above if clause.
/* 50-                        */            // thus implicitParamsStrMap needs to be initialized first
/* 51-                        *//* ...  */
/* 58-                        *//* ...  */
/* 59-                        */
/* 60-                        */    @Override
/* 61-                        */    public AccountSettings updateAccountSettings(Integer trendlocationWoeid,
/*   -                        *//*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/* 62-FormalParameterNamin(2)*/                                                 Boolean sleep_timeEnabled, String start_s
/* 63-FormalParameterNamin(2)*/                                                 String end_sleepTime, String time_zone, S
/* 64-                        */            throws TwitterException {
/* 65-                        */        List<HttpParameter> profile = new ArrayList<HttpParameter>(6);
/* 66-                        */        if (trendlocationWoeid != null) {
/*   -                        *//* ...  */
/* 81-                        *//* ...  */
/* 82-                        */            profile.add(new HttpParameter("lang", lang));
/* 83-                        */        }
/* 84-                        */        return factory.createAccountSettings(post(conf.getRestBaseURL() + "account/setting
/* 85-OptimizableToArrayCa    */                , profile.toArray(new HttpParameter[profile.size()])));
/* 86-                        */
/* 87-                        */    }
/* 88-                        */
/* 89-UnusedPrivateField      */    private int not_used = 0;
/* 90-                        */
/* 91-                        */}
```

Figure 8: Comparison between old and new version

3.1.4 Categorize the alerts

The alert in the old version is classified as **open** if there is an alert in the new version in the corresponding line with the same rule, same method name, and same variable name. The associated rule, the name of the method and variable affected by the alert are provided by PMD as part of the description of the alert. Otherwise, the alert is categorized as **fixed**.

Table 6 shows the classification of the alerts in the old version. For alerts 1 and 4, it is not possible to find an alert with the same rule, method and variable name in the related line of code in the new version. So, these alerts are classified as **fixed**.

For the remaining alerts, it is possible to find an alert with the same rule, method and variable name in the related line of code in the new version. So, these alerts are classified as **open**.

Marcio: referenciar as tabelas 4 e 5 no corpo do texto. Bruno: Tá referenciado, não? Botei em vermelho.

Table 6: Classifications of the alerts in the old version

id	line	rule	class	method	variable	idnew	linenew	category
1	3	UnusedImports	TwitterImpl	No method	No variable	NA	NA	Fixed
4	61	FormalParameterNamingConventions	TwitterImpl	updateAccountSettings	trend_locationWoeid	NA	NA	Fixed

Table 7 shows the classification of the alerts in the new version. The alert 8 is the only one for which there is no alert with the same characteristics in a line of the old version that is mapped to the original line in the new version. So it is the only **new** alert.

Table 7: Classifications of the alerts in the new version

id	line	rule	class	method	variable	idold	lineold	category
1	5	UnusedPrivateField	TwitterImpl	No method	logger	2	7	Open
2	47	OptimizableToArrayCall	TwitterImpl	TwitterImpl	No variable	3	48	Open
3	62	FormalParameterNamingConventions	TwitterImpl	updateAccountSettings	sleep_timeEnabled	5	62	Open
4	62	FormalParameterNamingConventions	TwitterImpl	updateAccountSettings	start_sleepTime	6	62	Open
5	63	FormalParameterNamingConventions	TwitterImpl	updateAccountSettings	end_sleepTime	7	63	Open
6	63	FormalParameterNamingConventions	TwitterImpl	updateAccountSettings	time_zone	8	63	Open
7	85	OptimizableToArrayCall	TwitterImpl	updateAccountSettings	No variable	9	85	Open
8	89	UnusedPrivateField	TwitterImpl	No method	not_used	NA	NA	New

3.2 Matches using the Abstract Syntax Tree and the relation between lines of code of the versions

In this algorithm we use the AST to create features that help to infer if the alerts in different versions must be considered the same.

In the Section 3.2.1, we show how the features are created.

Even though it's not possible to be sure if a pair alerts in different versions are same alert, we think that these features can provide some clues.

The features can be an input to a heuristic or a machine learning tool which can decide if the alerts are the same.

3.2.1 Features engineering

In this Section, we will consider the old version in Figure 9 and the new version in Figure 10

```
/*  1-                                  */package pack_x;
/*  2-                                  */
/*  3-                                  */import importX.function;
/*  4-                                  */
/*  5-                                  */class ClassX extends ClassY implements InterfX {
/*  6-                                  */    private long fieldX;
/*  7-                                  */
/*  8-                                  */    ClassX(int paramX, double paramY) {
/*  9-                                  */        int varX = function(paramX, paramY);
/* 10-                                  */        if (varX == 0)
/* 11-ControlStatementBraces            */            this.fieldX = 1;
/* 12-                                  */        else{
/* 13-                                  */            this.fieldX = 0;
/* 14-                                  */     }
/* 15-                                  */    }
/* 16-                                  */    @Override
/* 17-                                  */    public int methodX(int paramW, Boolean paramZ)
/* 18-                                  */    {
/* 19-                                  */        if (paramZ)
/* 20-ControlStatementBraces            */            fieldX = paramW;
/* 21-                                  */        else{
/* 22-                                  */            fieldX = 0;
/* 23-                                  */     }
/* 24-                                  */        return paramW + this.fieldX;
/* 25-                                  */     }
/* 26-                                  */}
```

Figure 9: Old version

```
/*  1-                                  */package pack_x;
/*  2-                                  */
/*  3-                                  */import importX.function;
/*  4-                                  */
/*  5-                                  */class ClassX extends ClassY implements InterfX {
/*  6-                                  */    private long fieldX;
/*  7-                                  */
/*  8-                                  */    ClassX(int paramX, double paramY) {
/*  9-                                  */        int varX = function(paramX, paramY);
/* 10-                                  */        if (varX == 0)
/* 11-                                  *//* ...  */
/* 12-                                  */            this.fieldX = 1;
/* 13-                                  */        }
/* 14-                                  */        else{
/* 15-                                  */            this.fieldX = 0;
/* 16-                                  */     }
/* 17-                                  */    }
/* 18-                                  */    @Override
/* 19-                                  */    public int methodX(int paramW, Boolean paramZ)
/* 20-                                  */    {
/* 21-                                  */        if (paramZ)
/* 22-ControlStatementBraces            */            fieldX = paramW;
/* 23-                                  */        else{
/* 24-                                  */            fieldX = 0;
/* 25-                                  */     }
/* 26-                                  */        return paramW + this.fieldX;
/* 27-                                  */     }
/* 28-                                  */}
```

Figure 10: New version

In Figure 12 we can see both ASTs, from the old and the new version. In this figure, the number of the nodes are meaningless. We can see the alerts linked to its nodes.

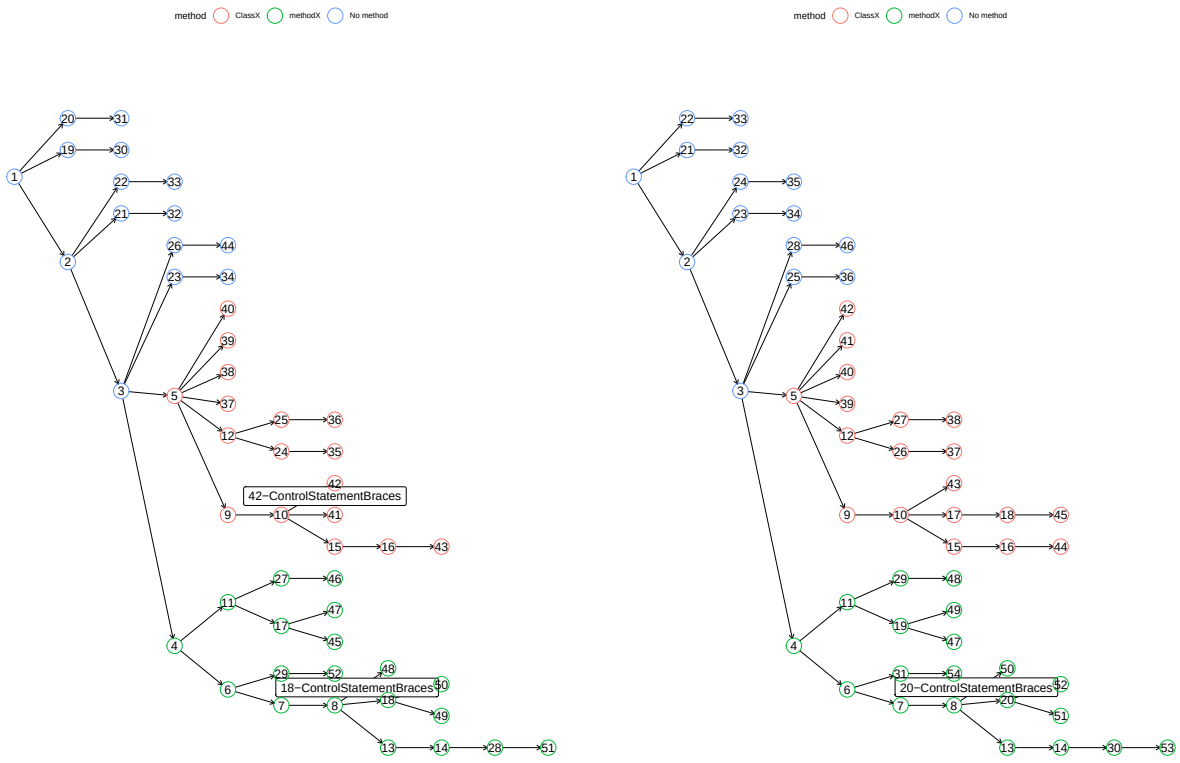

Figure 11: Abstract Syntax Trees. New and old versions

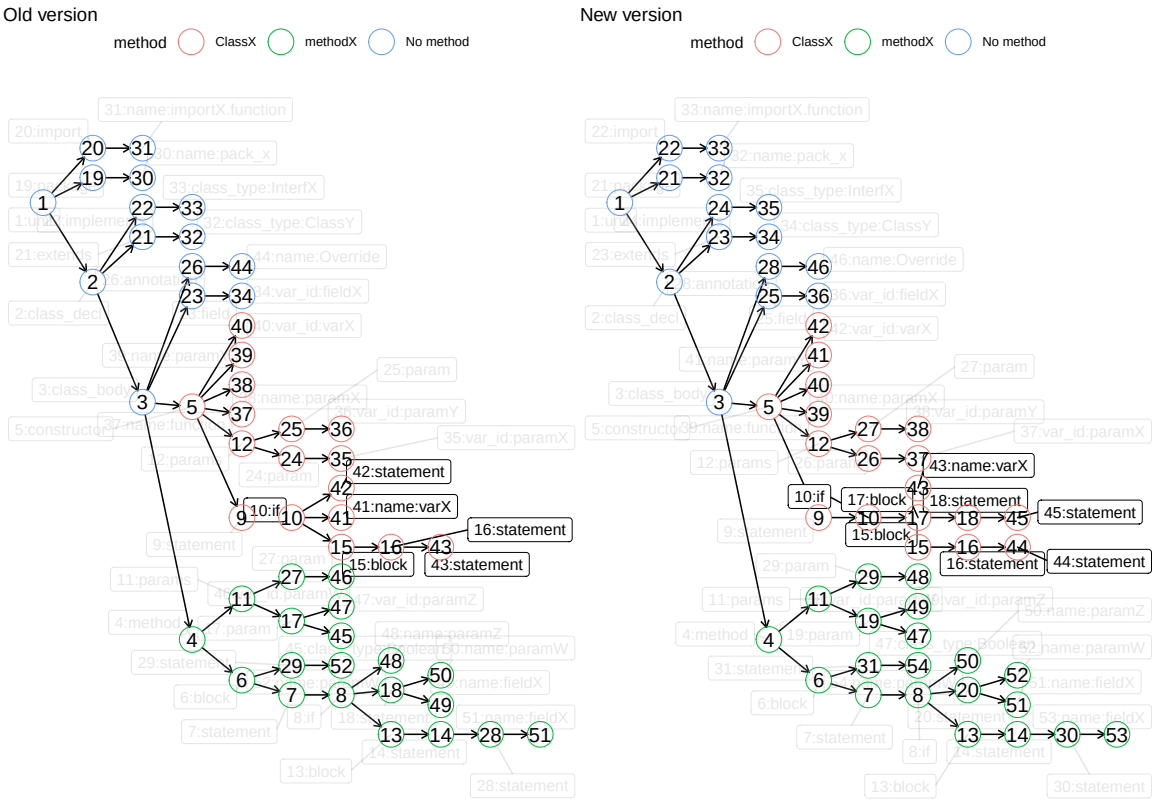

Figure 12: Abstract Syntax Trees compared. New and old versions

In Figure 13 the nodes with the same numbers defined as equivalent. We define that a node from an AST is equivalent to a node of the AST of the subsequent version if:

- Considering the relation between the lines of the two versions constructed as we saw in Section 3.1.3, the nodes in both trees begin and end in related lines
- The nodes are of the same kind

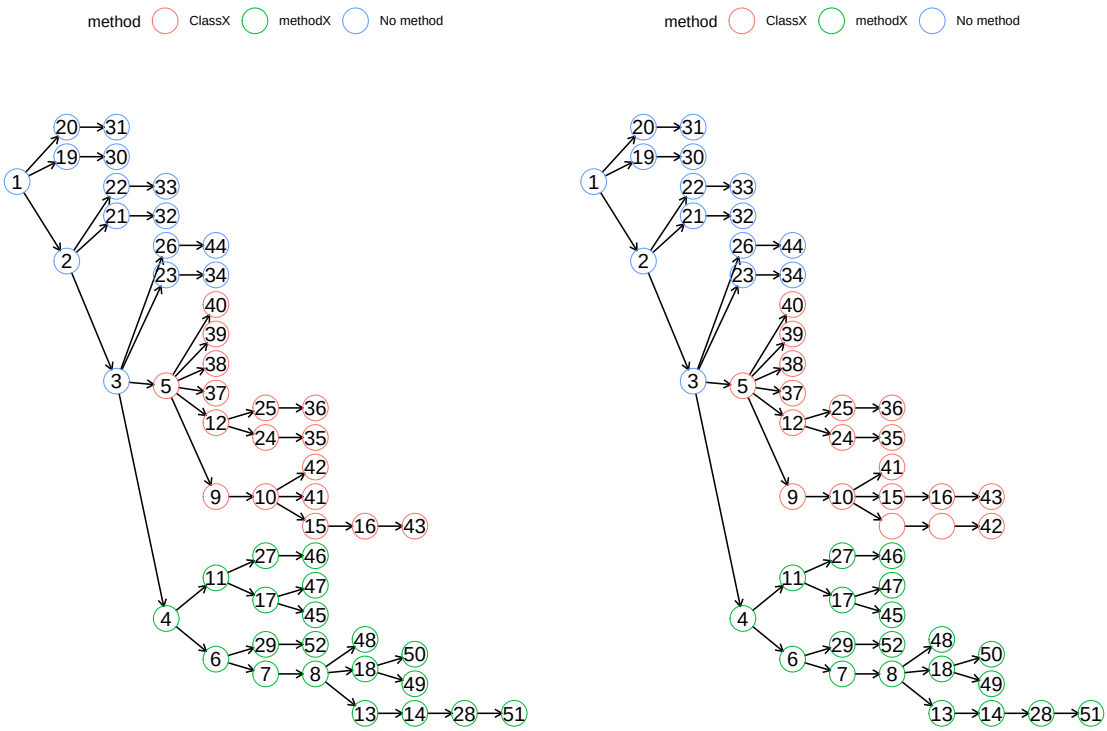

Figure 13: Abstract Syntax Tree

In Figure 14 we add the alerts that are related with nodes in the AST.

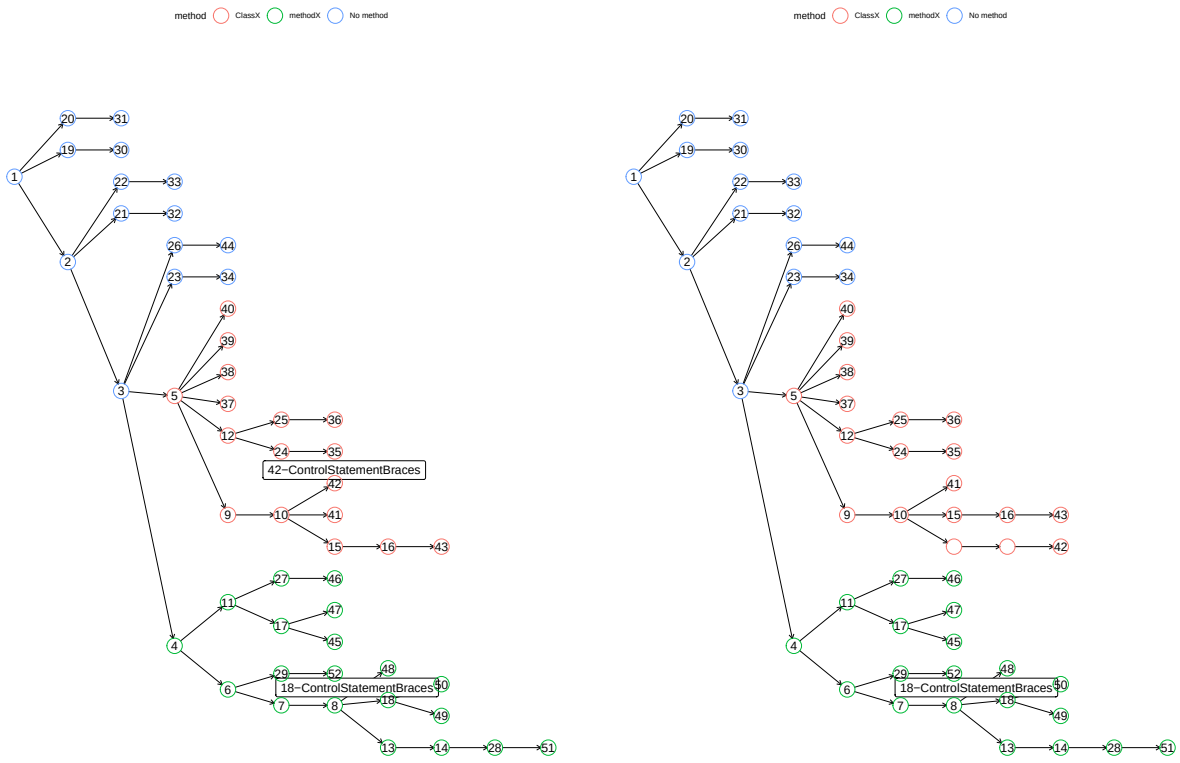

Figure 14: Abstract Syntax Tree

The path between the alert and the root of the AST can be seen in 15

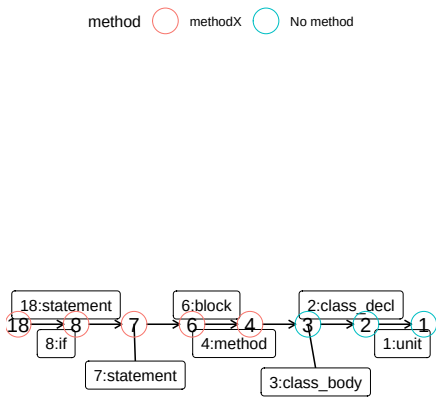

Figure 15: Abstract Syntax Tree

The algorithm generates a set of features for each pair of alerts (n, o) with one element n coming from the old version and one element o coming from the new version.

We propose the following list of features:

- Same Group ID: a boolean indicator that tells if the alerts are equivalent as defined in Figure 13
- Same Method: a boolean indicator that tells if the alerts belong to the same method. We know the alert's method following the path from the alert´s node to the root. The first node of the kind "method" found in this path defines the alert's method. If this is the same for o and for n, then they belong to the same method.
- Same Block: a boolean indicator that show if the o and n belong to the same block. It is defined the same way the "Same method" indicator is defined.
- Line distance: o and n have a begin line $b(o)$ and $b(n)$ and an end line $e(n)$ and $e(n)$. Line distance is $abs(mean(b(o), e(o)) - mean(b(n), e(n)))$
- Normalized line distance: this is the line distance but normalized by the size of the last common node.

Table 8 shows the combinations (n, o) in the example. There are $2 \cdot 1 = 2$ combinations whereas we have two alerts in the old version and one alert in the new one. The alert ids are the original ones, shown in Figure 11.

Table 8: Classifications of the alerts in the old version

Alert id (new version)	Alert id (old version)	Same Group	Same Method	Same Block	Line Distance	Line Distance normalized by block)	Line Distance normalized by block)	Line Distance normalized by complie unit)
20	18	TRUE	TRUE	TRUE	0.00	0.00	0.00	0.00
20	42	FALSE	FALSE	FALSE	10.00	0.43	0.37	0.37