GENERACIÓN DE UN ALGORITMO PARA EL ANÁLISIS DE SIMILITUDES DE CÓDIGO FUENTE EN LENGUAJES JAVA Y PYTHON

HÉCTOR JOSÉ CALDERÓN PACHÓN

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ D.C.
2019

GENERACIÓN DE UN ALGORITMO PARA EL ANÁLISIS DE SIMILITUDES DE CÓDIGO FUENTE EN LENGUAJES JAVA Y PYTHON

HÉCTOR JOSÉ CALDERÓN PACHÓN

Proyecto integral de grado para optar al título de
INGENIERO DE SISTEMAS

Asesor
CHRISTIAN CAMILO APARICIO BAQUEN
MEng. Seguridad de la Información

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ D.C.
2019

CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

Pág.

RESUMEN

El propósito de este documento es generar el reporte del proyecto de grado. Su desarrollo está basado en la búsqueda e implementación de un algoritmo que permita mecanismos para encontrar plagio en código fuente. Este, debe considerar los requerimientos que posee la Universidad de los Andes.

Adicionalmente, el funcionamiento de este algoritmo se encuentra dentro de una aplicación web, siendo ejecutado luego de cargar varios proyectos en una carpeta específica. Al completar la carga, se inicia la comparación individual de cada fichero, dando como resultado los valores necesarios para encontrar el grado de relación entre este y los otros documentos del mismo nombre. Con el resultado de la comparación, la persona que carga la información determina si hay o no casos de plagio en el reporte que el algoritmo genera.

Teniendo esto en cuenta, el proyecto se lleva a cabo con la motivación de brindar capacidades y mecanismos a los profesores para enfrentar de forma eficiente y automática el plagio. Por consiguiente, se centra en generar nuevos mecanismos de acción contra esta práctica. Para lograr este objetivo, se describe a continuación el esquema del presente documento.

En este, se presenta en 6 secciones el proceso realizado. Inicialmente, en la sección **Descripción general**, se definen los objetivos del proyecto y se da una aproximación a diversas soluciones que se han detallado hasta el momento. Se identifican las necesidades específicas para los lenguajes de programación Java y Python definiendo el problema y su importancia. Luego, se encuentra el **Diseño y especificaciones**, donde se define el problema que se desea abordar, con los requerimientos y restricciones que aplican para el mismo.

El siguiente punto es el **Desarrollo del diseño**, en el cual se detalla la información que se consultó para realizar el diseño y posteriormente la **Implementación**. En esta sección, se presenta el proceso que se llevó a cabo para completar el desarrollo. Finalmente, se muestran los **Resultados** y las **Conclusiones** del trabajo desarrollado, exponiendo el nivel de completitud que se consiguió del proyecto y el trabajo a futuro para la idea de proyecto.

ABSTRACT

The main purpose of this document is to generate the report of the degree project. Its development is based on the search and implementation of an algorithm that provides mechanisms to detect plagiarism in main codes. This algorithm must take into account the requirements specified by Universidad de los Andes.

In addition, this algorithm operates in a web application, as well as executed after adding various projects in an specific file. When the upload is completed, an individual comparation of each file is started; as a result, it communicates the necessary values in order to find the degree of relation between this and other files of the same name. With this information, the operator can determine whether or not the case can be considered as plagiarism.

Considering all of the above, the project is inspired by the wish to provide teachers capacities and mechanisms to face, in an automatic and efficient way, plagiarism. Consequently, it is centered in generating new action mechanisms against this practice. In order to achieve this objective, a description of this document is presented below.

In this document are six sections containing the process. Initially, in the section of **General Description**, objectives are defined as well as an approach to different solutions previously detailed. Specific necessities for Java and Python languages are specified, defining the main problem and its importance. Afterwards, **Design and Specifications** defines the target problem, alongside with requirements and restrictions that apply to the case.

Next section, **Design Development**, details the information consulted so as to make the design and **implementation**. In this section, the development of this whole process is presented. Finally, **Results** and **Conclusions** of the present worked expose the level of completeness that was achieves in this project; as well as future work needed for this project.

1. INTRODUCCIÓN

Los estudiantes, a lo largo de su vida académica deben realizar trabajos y proyectos, de forma individual o colectiva, asociados a diversas materias. Al desarrollar estas actividades, los alumnos deben ser conscientes de que toda producción que entreguen a título personal se refiere a su propia autoría, pues sustentan el pensamiento y la capacidad individual del autor. En este sentido, encontrar similitudes entre diferentes trabajos, representa el desarrollo de ideas colectivas o el uso de fuentes de información de terceros. Si la información no se encuentra citada y referenciada de las fuentes que se han usado, esta se identificaría como plagio.

El plagio se puede considerar como la acción de apropiarse del material intelectual de otros (RAE, 2018). Este comportamiento es una realidad que debe ser enfrentada por las instituciones educativas, ya que, representan un hecho que va en contra de los deberes que posee un estudiante. Para esto, existen algunos mecanismos que abarcan los casos de fraude más generales, los textos escritos. Uno de ellos, consiste en comparar lo entregado por un estudiante y una base de datos. En esta, se encuentran almacenados documentos que otros estudiantes han realizado anteriormente y que no incurrieron en copia. Si se encuentra similitud con uno o más archivos y no poseen las respectivas referencias del contenido que parece no ser de autoría propia, se comprueba el caso de plagio.

Otro mecanismo, por ejemplo, consiste en realizar la comparación entre la información entregada por estudiantes de la misma clase y la información que se han presentado estudiantes en años anteriores y contextos o espacios diferentes. En este caso, se comprueba la imitación para todos los estudiantes involucrados que posean parte del documento en común.

Sin embargo, el proceso para encontrar este tipo de fraude es sencillo, no obstante, llevarlo a cabo suele consumir mucho tiempo y dedicación por parte de las personas que se encargan de realizar el proceso. Por lo tanto, las instituciones académicas en su deber de garantizar una educación de calidad para sus estudiantes han de disponer recursos de la institución para la prevención, detección y análisis de posibles casos. Cuando son instituciones muy grandes, se tienen que implementar estrategias que permitan minimizar el uso de estos recursos, pues puede presentarse que no se disponga de todos los elementos necesarios.

Al considerar estas posibilidades, se encuentra que las materias más afectadas son las del ciclo básico, donde los estudiantes que tienen este tipo de prácticas pueden pasar desapercibidos entre los alumnos que presentan dificultades en el aprendizaje y son dedicados. Para combatir este fenómeno, es vital contar con una cobertura en esta modalidad que, además de ser efectiva con los recursos disponibles, permita tener un mayor control del aprendizaje y métodos de los estudiantes.

Actualmente, la Universidad de los Andes dentro de la plataforma digital de clases “Blackboard”, posee una herramienta llamada SafeAssign. Esta, se habilita cada vez que los estudiantes suben documentos escritos, siendo conscientes que su trabajo será comparado con el banco de textos en búsqueda de plagio. Adicionalmente, en el último semestre (segundo semestre del 2018) se ha incorporado el uso de la herramienta Turnitin, un software privado que la universidad está utilizando como medida preventiva para detectar plagio en escritos.

El uso de Turnitin se realiza de manera libre, la finalidad de este recurso es darle oportunidad al estudiante de validar que su trabajo para denotar que supera el filtro de plagio. Este último paso conlleva a la entrega del documento sin consecuencias. Sin embargo, si Turnitin encuentra un caso de plagio, le permite al individuo reestructurar su composición antes de la entrega oficial, que es comprobada por ambos softwares, SafeAssign y Turnitin, según decisión del profesor.

A pesar de la existencia de herramientas como SafeAssign o Turnitin, detectar el plagio en el código fuente es más complejo, ya que estos softwares no poseen una funcionalidad con lenguajes formales, pues de usar los mismos mecanismos, la sintaxis del lenguaje sería responsable del mayor porcentaje de ocurrencia de código con similitud. Para comprobar el plagio en estos escenarios, existen proyectos como “Medida de la similitud de software” (MOSS, por sus siglas en inglés) que permite realizar la evaluación de plagio en código fuente.

Al explorar soluciones como MOSS, se evidencia que los softwares existentes no suplen las necesidades que posee la universidad por ciertas restricciones o interfaces que implementan, por consiguiente, esto causa que el proceso se convierta en una actividad manual. Por ejemplo, en el semestre 2018-10 en la Universidad de los Andes, el equipo de Cupi2 comparó los trabajos finales de todos los estudiantes del curso APO I, llevando a cabo una labor de más de dos semanas y dejando como resultado al menos ciento cincuenta (150) estudiantes con procesos disciplinarios abiertos.

Al analizar diferentes algoritmos para este fin, se obtuvo uno que cumplía con los requisitos no funcionales que se buscaban. Este, produce un resultado que muestra los archivos en forma de sus partes más importantes, validando que estos no hagan parte de un archivo en común, llamado esqueleto. Estos valores se conocen como fingerprints[1], los cuales representan la identidad única de cada documento. Si dos o más archivos poseen al menos uno de sus fingerprints iguales se dice que existe un posible caso de copia entre estos.

[1] Huella dactilar en inglés.

2. DESCRIPCIÓN GENERAL

2.1 OBJETIVOS

Los objetivos enunciados a continuación permitieron definir los límites y el alcance de la solución propuesta.

2.1.1 OBJETIVO GENERAL

Generar un algoritmo para el análisis de similitudes en código fuente en lenguajes Java y Python, realizado por un estudiante, teniendo en cuenta la necesidad de omitir un código en común que haya sido entregado como base a los estudiantes.

2.1.2 OBJETIVOS ESPECIFICOS

- Comprender las tecnologías a utilizar.
- Analizar los algoritmos de comparación, enfocándose en aquellos orientados a datos replicados.
- Realizar la comparación entre archivos.
- Comprender técnicas de minificación[2] y refactorización.
- Analizar los requerimientos de funcionalidad del aplicativo.
- Priorizar la comparación exitosa entre archivos omitiendo un texto en común.

[2] La minificación es el proceso que elimina datos innecesarios para mejorar el desempeño del código fuente, disminuyendo su tamaño de almacenamiento, sin afectar el correcto funcionamiento del mismo.

2.2 ANTECEDENTES

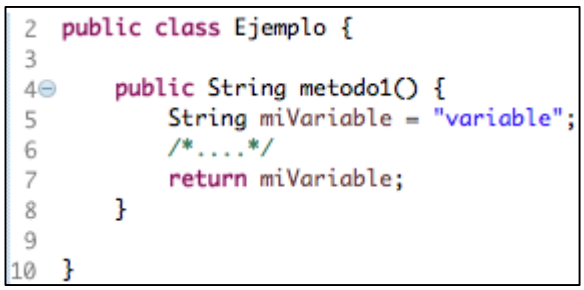
A continuación, se presentan los conocimientos relevantes para el correcto entendimiento e interpretación de la solución encontrada.

2.2.1 PLAGIO EN CÓDIGO

El plagio en código es el uso o modificación del código de un tercero presentado como código propio. Este tipo de plagio se puede mostrar de diversas formas, incluyendo la acción de copiar y pegar información. Algunas de ellas se presentan a continuación:

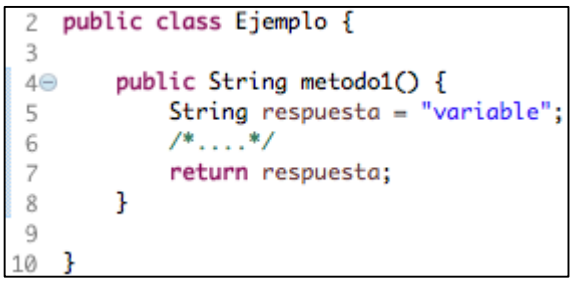
1. **Cambiar el nombre de una variable.** Es un procedimiento simple donde la persona toma el código de una fuente externa y para hacerlo parecer propio, decide cambiar el nombre de las variables presentes.

Figura 1. Ejemplo 1 del cambio de nombre de una variable



Fuente: Autor. Código fuente original en ECLIPSE

Figura 2. Plagio del ejemplo 1 de cambio de nombre de una variable



Fuente: Autor. Código fuente modificado en ECLIPSE

Este tipo de plagio se realiza con la intención de confundir a la persona que revisa manualmente el código. Si esta no hace una revisión precisa, puede

dejar pasar como originales estructuras que son idénticas. Por ejemplo, las imágenes anteriores poseen nombres de variables diferentes.

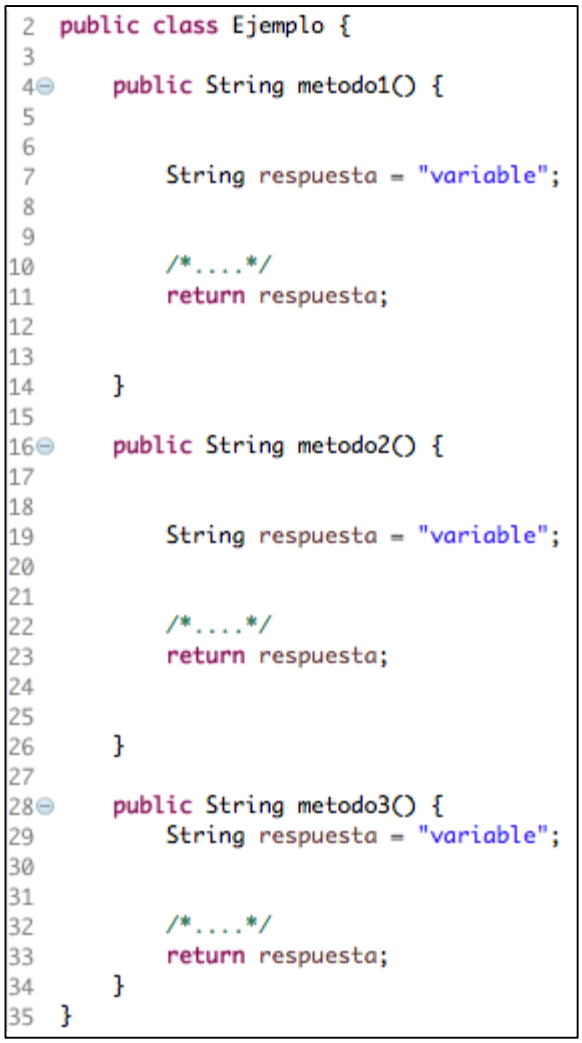
2. **Generar más espacios o tabulaciones entre los elementos**. De forma similar a la anterior técnica, es un procedimiento sencillo que permite engañar a la persona que revisa el código, pues la forma en la cual este se realiza puede ser apreciada de manera diferente dependiendo del espacio interlineado.

Figura 3. Ejemplo para generar más espacios o tabulaciones entre los elementos.

```
2  public class Ejemplo {
3
4      public String metodo1() {
5          String respuesta = "variable";
6          /*....*/
7          return respuesta;
8      }
9
10     public String metodo2() {
11         String respuesta = "variable";
12         /*....*/
13         return respuesta;
14     }
15
16     public String metodo3() {
17         String respuesta = "variable";
18         /*....*/
19         return respuesta;
20     }
21 }
```

Fuente: Autor. Código fuente original en ECLIPSE

Figura 4. Plagio del ejemplo para generar más espacios o tabulaciones entre los elementos.



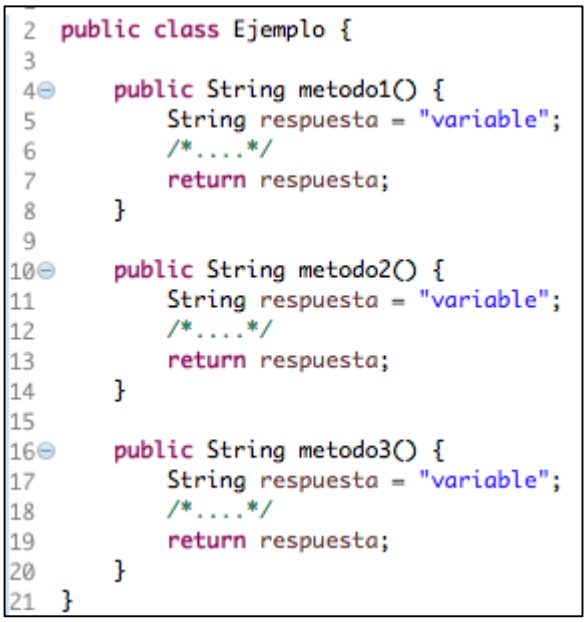
```
2  public class Ejemplo {
3
4      public String metodo1() {
5
6
7          String respuesta = "variable";
8
9
10         /*....*/
11         return respuesta;
12
13
14     }
15
16     public String metodo2() {
17
18
19         String respuesta = "variable";
20
21
22         /*....*/
23         return respuesta;
24
25
26     }
27
28     public String metodo3() {
29         String respuesta = "variable";
30
31
32         /*....*/
33         return respuesta;
34     }
35 }
```

Fuente: Autor. Código fuente modificado en ECLIPSE

Dependiendo de la forma en que el archivo es modificado, se puede evidenciar el plagio a simple vista. Por ejemplo, en la ilustración anterior, se puede apreciar que el interlineado es irregular, sin un patrón.

3. **Modificar el orden del archivo**. Es efectivo en archivos con una gran cantidad de métodos y líneas de código, pues cambia la forma en la cual se ve el conjunto de elementos.

Figura 5. Ejemplo de la modificación del orden del archivo

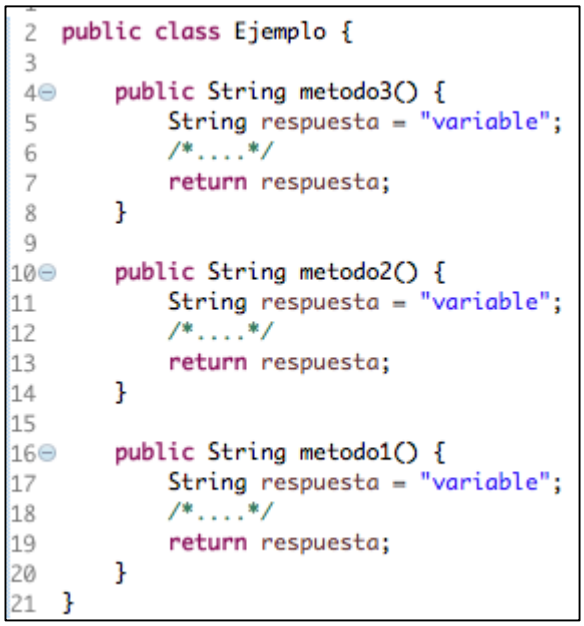


```
2  public class Ejemplo {
3
4      public String metodo1() {
5          String respuesta = "variable";
6          /*....*/
7          return respuesta;
8      }
9
10     public String metodo2() {
11         String respuesta = "variable";
12         /*....*/
13         return respuesta;
14     }
15
16     public String metodo3() {
17         String respuesta = "variable";
18         /*....*/
19         return respuesta;
20     }
21 }
```

Fuente: Autor. Código fuente original en ECLIPSE

En archivos pequeños, este tipo de modificaciones no consiguen su objetivo, ya que el cambio no es suficientemente significativo para engañar el ojo humano. Sin embargo, en documentos de más de 200 líneas de código la situación permite pasar desapercibido esta suplantación, ya que no se consigue tener una perspectiva completa del archivo.

Figura 6. Plagio del ejemplo de la modificación del orden del archivo

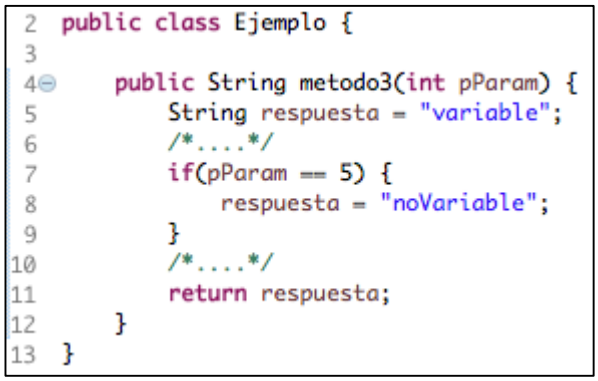


```
2  public class Ejemplo {
3
4      public String metodo3() {
5          String respuesta = "variable";
6          /*....*/
7          return respuesta;
8      }
9
10     public String metodo2() {
11         String respuesta = "variable";
12         /*....*/
13         return respuesta;
14     }
15
16     public String metodo1() {
17         String respuesta = "variable";
18         /*....*/
19         return respuesta;
20     }
21 }
```

Fuente: Autor. Código fuente modificado en ECLIPSE

4. **Añadir o modificar estructuras de control.** Las estructuras de control son aquellas que modifican el curso de acciones de un programa en ejecución. Estas, determinan si en un momento se debe tomar un camino u otro, dependiendo del estado de la aplicación en ese instante.

Figura 7. Añadir o modificar estructuras de control

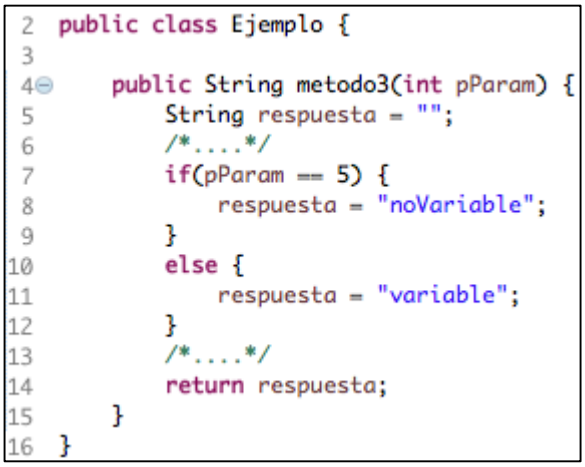


```
2  public class Ejemplo {
3
4      public String metodo3(int pParam) {
5          String respuesta = "variable";
6          /*....*/
7          if(pParam == 5) {
8              respuesta = "noVariable";
9          }
10         /*....*/
11         return respuesta;
12     }
13 }
```

Fuente: Autor. Código fuente original en ECLIPSE.

En este ejemplo, el código original muestra una estructura de control que cambia el mensaje que va a salir y depende del parámetro que entra.

Figura 8. Plagio de añadir o modificar estructuras de control



```
2  public class Ejemplo {
3
4      public String metodo3(int pParam) {
5          String respuesta = "";
6          /*....*/
7          if(pParam == 5) {
8              respuesta = "noVariable";
9          }
10         else {
11             respuesta = "variable";
12         }
13         /*....*/
14         return respuesta;
15     }
16 }
```

Fuente: Autor. Código fuente modificado en ECLIPSE

El plagio se presenta al copiar y modificar el código el cual añade una estructura que permite dar una respuesta igual y deja la implementación en diferente forma.

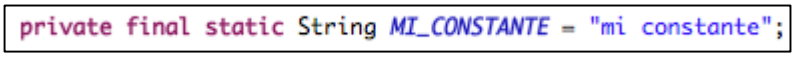
Entre estas formas de plagio, la que presenta mayor dificultad para ser detectada es la última, puesto que cambia la forma lógica expresada, aunque el resultado sea el mismo. Para este proyecto, no se tendrá en cuenta esta forma de plagio, pues a todos los estudiantes se les entrega el mismo enunciado o código esqueleto. Este tipo de comprobación implica un mecanismo avanzado para entender y diferenciar entre lo que es código original y código copiado en diferentes versiones de un mismo archivo.

Adicionalmente, es importante tener en cuenta que, si el estudiante opta por hacer plagio de este tipo, le implicará mayor trabajo, puesto que generar una premisa original facilita la producción lógica, mientras que al suplantar ideas y códigos debe evitar generar alertas de plagio, entorpeciendo el desarrollo del mismo.

2.2.2 SINTAXIS DEL LENGUAJE

Los lenguajes de programación poseen una sintaxis que permite traducir las ideas en implementaciones. Depende del lenguaje de programación, esta sintaxis puede tener una cadena de caracteres ampliada o reducida para describir una funcionalidad. Por ejemplo, en el lenguaje Java para escribir una constante hace falta escribir:

Figura 9. Ejemplo de constante en Java



Fuente: Autor. ECLIPSE.

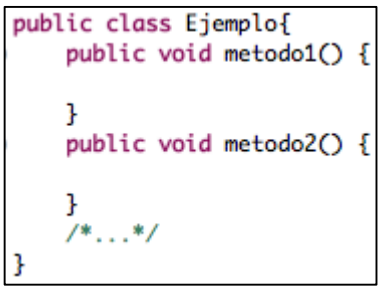
Mientras que, en Python, la misma interpretación de una constante se hace por medio del siguiente código:

Figura 10. Ejemplo de constante en Python

Fuente: Autor. NOTEPAD++

Con los ejemplos anteriormente expuestos, se desea mostrar que para los lenguajes existen reglas que pueden o no aplicar estrictamente. Por ejemplo, para Java una norma sintácticamente fuerte es que los fragmentos de código deben estar encapsulados entre corchetes, indicando que esas líneas de código pertenecen a un objeto o acción específico.

Figura 11. Ejemplo de corchetes en Java



Fuente: Autor. ECLIPSE

Adicionalmente, el final de cada sentencia debe terminar con punto y coma, siendo esta otra norma sintácticamente fuerte.

Figura 12. Ejemplo de final de línea en Java

```
public String metodo1() {
    String respuesta = "variable";
    /*....*/
    return respuesta;
}
```

Fuente: Autor. ECLIPSE

A partir de esta regla Java recomienda aplicar, entre otras, la regla de la identación[3] y salto de línea, para ver más elegante y legible el código escrito.

Figura 13. Ejemplo de código que no sigue las reglas sintácticas Java

```
public class Ejemplo {
public String metodo1() {String respuesta = "variable";/*....*/while(respuesta.length()>1) {respuesta = respuesta.substring(0, respuesta.length()-1);}/*....*/return respuesta;
```

Fuente: Autor. ECLIPSE

El código anterior funciona perfectamente, pero es ilegible. Para mejorar esto, siguiendo las recomendaciones del lenguaje, se dividirá con saltos de línea al terminar una instrucción y con identación.

Figura 14. Ejemplo de código siguiendo las reglas sintácticas Java

```
public class Ejemplo {

    public String metodo1() {
        String respuesta = "variable";
        /*....*/

        while(respuesta.length()>1) {
            respuesta = respuesta.substring(0, respuesta.length()-1);
        }

        /*....*/
        return respuesta;
    }

}
```

Fuente: Autor. ECLIPSE

Por otro lado, en Python, el orden del código posee solo una norma sintácticamente fuerte, la identación, que para el lenguaje natural es más intuitivo que las reglas implementadas en Java.

[3] El término es un anglicismo acuñado en la programación. Se refiere a la misma sangría.

Figura 15. Ejemplo de regla sintáctica Python

```
while ans:
    question = raw_input("Hola! coloca un Si o un No: (press enter to quit) ")
    answers = random.randint(1,8)
    if question == "":
        sys.exit()
    elif answers == 1:
        if question == "Si":
            #do something
            print "muy bien! Estoy en otro nivel de identación, es decir estoy dentro del if"
        else:
            print"Aw, fue negativo"
        print "Estoy en el mismo nivel de identación que el if"
    elif answers == 2:
        print "Estoy dentro del elif"
    print "Estoy dentro del while"
print "Estoy fuera de todo."
```

Fuente: Autor. NOTEPAD++

Finalmente, con este conocimiento se puede abordar un algoritmo de detección de similitudes, considerando que las ocurrencias no pueden ser coincidencias con la sintaxis, convirtiéndose estos en falsos positivos.

2.2.3 PALABRAS RESERVADAS DEL LENGUAJE

La sintaxis del lenguaje se define principalmente por sus palabras reservadas. En ocasiones estas palabras cambian su forma, pero su utilidad será la misma. Por ejemplo, un *elif* de python es un *else if* de Java. Para reconocerlas, basta con tener en el IDE[4] o en el editor de texto, un linter[5] que conozca el lenguaje, de esta forma cambiará el color de los diferentes elementos del código, donde las palabras reservadas se pintarán en color morado.

Figura 16. Ejemplos de palabras reservadas en Java y Python

Fuente: Autor. ECLIPSE y NOTEPAD++.

[4] Ambiente de desarrollo. Software especializado para facilitar el desarrollo en un lenguaje de programación.
[5] Accesorio de algunos editores de texto que están observando constantemente la sintaxis para un lenguaje de programación.

En las anteriores ilustraciones se pueden ver palabras reservadas. En fondo blanco, se muestran palabras en el lenguaje Java. En fondo negro, se observan palabras en el lenguaje Python.

Las palabras reservadas de Java se han mantenido constantes a través de las diversas versiones. A continuación, se presentan estas palabras.

Tabla 1. Palabras reservadas de Java

abstract	assert	boolean	break	byte
case	catch	char	class	const
catch	char	do	double	else
enum	extends	final	finally	float
for	goto	If	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

Fuente: Autor. Palabras reservadas de Java.

Por el contrario, Python posee diferencias de su segunda versión a la tercera. A continuación, se presentan las palabras reservadas en Python 2:

Tabla 2. Palabras reservadas en Python

and	as	assert	break	class
continue	def	del	elif	else
except	exec	finally	for	from
global	if	import	in	is
lambda	not	or	pass	print
raise	return	try	while	with
yield				

Fuente: Autor. Palabras reservadas de Python 2.

Para Python 3 las palabras **exec** y **print** dejan de ser reservadas. Como adición, entran a la lista las siguientes:

Tabla 3. Palabras reservadas en Python3

nonlocal	True	False	none

Fuente: Autor. Palabras reservadas añadidas en Python 3

2.2.4 PROCESO DE MINIFICACIÓN

La minificación es el proceso de disminuir el tamaño de un archivo, retirando caracteres que no son relevantes y ofuscando el contenido de este. Se usa principalmente como garante de seguridad para la lectura del código fuente de un programa, o como medida que permite una eficiencia superior al ejecutar la aplicación.

Por ejemplo, si se quisiera minificar el siguiente fragmento de código:

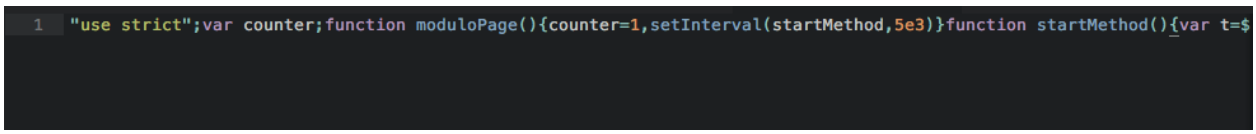
Figura 17. Ejemplo 1 de minificación en JavaScript

```
"use strict";

var counter;

function moduloPage(){
    var controler=controler||{};
    counter = 1;
    setInterval(startMethod,5000);
};

function startMethod(){
    var opacityDelay = 50;
    var opacityChange = 0.05;
    var imagen = $("#background");
    var cambiar = true;

    var mostrar = function(opacity){
        opacity = opacity + opacityChange;
        imagen.css('opacity', opacity);
        if (opacity>= 1){
            return;
        }
        setTimeout(function(){
            mostrar(opacity);
        }, opacityDelay);
    };

    var desvanecer = function(opacity){
        opacity = opacity - opacityChange;
        imagen.css('opacity', opacity);
        if (opacity<=0){
            imagen.trigger('mostrar');
            return;
        }
        setTimeout(function(){
            desvanecer(opacity);
        }, opacityDelay)
```

Fuente: Autor. NOTEPAD++

El resultado sería:

Figura 18. Ejemplo 2 de minificación en JavaScript



Fuente: Autor. NOTEPAD++

De esta forma, el código pierde singularidades que no son relevantes para su ejecución, aunque se vuelve ilegible para las personas.

2.2.4 PROPIEDADES DE UN ALGORITMO DE DETECCIÓN

Se ha encontrado que para un algoritmo de detección de plagio se deben considerar las siguientes tres propiedades (Schleimer, Wilkerson, Aiken & Berkeley, (n.d)):

1. **Insensibilidad**: El algoritmo debe ser capaz de encontrar similitudes sin considerar cambios por letras en mayúsculas, por cantidad de espacios en blanco o nombre de variables.
2. **Supresión de ruido**: El descubrimiento de una similitud de cadena de carácteres entre dos archivos debe ser lo suficientemente grande para suponer que ha sido copiado, teniendo en cuenta la primera propiedad. Si la similitud en términos de cadena de caracteres es pequeña, no se debería aceptar, pues se puede inducir a que salgan falsos positivos. Por ejemplo, una similitud en la palabra “class” no tiene fundamentos en ser copia, dado que hace parte de la sintaxis del lenguaje.
3. **Independencia de posición**: El algoritmo debe ser capaz de entender, detallar y mostrar similitudes sin importar el lugar en que se encuentre la semejanza, capaz de encontrar paralelismos entre las primeras cadenas de caracteres de un documento con las últimas cadenas de caracteres de otro documento, por ejemplo.

Estas propiedades se definen como las características necesarias para poder garantizar la correcta implementación de un algoritmo de comparación en código fuente para encontrar similitudes (Thesis & Bostan, 2017). La primera propiedad descrita es, en esencia, la implementación existente en la mayoría de los algoritmos para encontrar similitudes en texto. Se basa en el proceso de minificación, el cual permite eliminar los caracteres que no son representativos y renombra variables de forma genérica, reversando las modificaciones que generan un cambio en la forma del código. Es decir, esta característica acoge directamente los tipos de plagio uno y dos.

La segunda propiedad, es encontrada en algoritmos de huella dactilar[6] de tamaño ***k***, los cuales permiten superar la sintaxis del lenguaje al trabajar con un mínimo de ***k*** caracteres que sean iguales. Esta característica permite encontrar similitud en lenguajes de programación, donde es la sintaxis la causa de la mayor aparición de similitudes si el número ***k*** es lo suficientemente pequeño.
Finalmente, la tercera propiedad implica un manejo de la ubicación de los caracteres en el documento. Para este propósito, se debe utilizar y modificar la forma en que un algoritmo de huella dactilar de tamaño ***k*** almacena la información, permitiendo añadir algún tipo de cabecera meta[7] que muestre la ubicación sin modificar el hallazgo de la huella dactilar.

2.2.5 ALGORITMOS DE HUELLA DACTILAR

Una subcadena de caracteres de longitud K se define como K-gram, donde la K significa el número de caracteres que posee la subcadena. Este término es acuñado para el análisis de todas las posibles subcadenas de caracteres que existan en una cadena dada (K-mer (n.d.) En Wikipedia).

EsteEsUnTexto
(a) Cadena de caracteres

EsteE steEs teEsU eEsUn EsUnT sUnTe UnTex nText Texto
(b) 5-gram subcadenas de texto

En la figura anterior se puede ver un ejemplo de K-gram sobre una cadena de caracteres. La cadena consta de 13 caracteres, y se define la K como 5. Para saber el número de K-gram que saldrá de la cadena de caracteres inicial, se debe hacer el cálculo del número de caracteres totales - K + 1. En este caso, serían 9 K-gram (13-5+1 = 9)

Estas subcadenas, o K-gram, son usadas ampliamente en los algoritmos de huella dactilar, ya que permiten lograr la supresión de ruido de manera satisfactoria (Schleimer, Wilkerson, Aiken & Berkeley, (n.d)). Con un K preciso, se pueden omitir similitudes que son falsos positivos dados por la sintaxis del lenguaje, sin descartar las semejanzas que se encuentren con un tamaño igual o superior al K.

2.2.5.1 Algoritmo de Karp-Robin

El algoritmo de Karp-Robin es considerado el primer algoritmo de huella dactilar (Schleimer, Wilkerson, Aiken & Berkeley, (n.d)). Este nace con la intención de encontrar la cantidad de veces una subcadena de caracteres ***S*** con longitud ***K***

[6] Conocidos como Algoritmos de Fingerprint, por su nombre en inglés.
[7] Un valor meta significa que es un valor que la aplicación debe conocer, pero es transparente al usuario.

aparece en una cadena de caracteres mucho más larga. El algoritmo se define como la comparación de los hashes[8] de todos los K-gram contra el hash de la cadena S.

Este procedimiento se realiza por medio de los hashes para permitir comparar cada subcadena con un número único. Obtener el hash de una cadena de caracteres se vuelve costoso a medida que la cadena se hace más larga. Esto quiere decir que, para un K muy grande, la eficiencia del algoritmo sería un déficit en la estrategia, pues consumiría mucho tiempo (Schleimer, Wilkerson, Aiken & Berkeley, (n.d)).

Debido a este inconveniente, Karp-Rabin utiliza recursivamente las operaciones que se realizan para el anterior hash al momento de generar el siguiente hash. Por ejemplo, para el K-gram C_1, C_2, C_3… C_k, donde C es el caracter en una posición de la cadena de caracteres, existe el hash del mismo, que se representa de la siguiente forma

$$H(K-gram) = H(C_1, C_2, C_3 \dots C_k)$$

Donde por cada carácter dentro del hash, existe un dígito en base b que lo acompaña (Schleimer, Wilkerson, Aiken & Berkeley, (n.d))

$$H(C_1, C_2, C_3 \dots C_k) = C_1 * b^{k-1} + C_2 * b^{k-2} \dots + C_k$$

Con el primer hash identificado, es posible encontrar el siguiente, teniendo en cuenta la fórmula que se presenta a continuación (Schleimer, Wilkerson, Aiken & Berkeley, (n.d)):

$$H(C_2, C_3, C_4 \dots, C_{k+1}) = (H(C_1, C_2, C_3 \dots C_k) - C_1 * b^{k-1}) * b + C_k + 1$$

Se retira siempre el primer carácter del valor del hash y se transforma el resto de los caracteres al valor que deberían tener antes de ser sumados al último carácter, que no posee un dígito en base b que lo transforme, se logra hacer una recursividad que con cuatro operaciones es más eficiente que el calcular k-1 multiplicaciones y k sumas por cada K-gram.

Aunque la estrategia posee una eficiencia superior, también tiene una debilidad. Este algoritmo de hash por la forma en la que está planteado, por cada nuevo elemento que se añade este no es significativo para variar el valor final de la función de hash. Para superar esta debilidad, se reestructura el algoritmo, aumentando una vez la base b para cada carácter e invirtiendo el orden de las operaciones aritméticas finales (Schleimer, Wilkerson, Aiken & Berkeley, (n.d)).

$$H'(C_1, C_2, C_3 \dots C_k) = C_1 * b^k + C_2 * b^{k-1} + \dots + C_k * b$$

[8] Un Hash es una función de picadillo, donde la entrada se transforma en algo que es único para el valor de entrada y que no es posible devolver desde su salida.

Nuevamente, con el primer hash encontrado, se puede calcular el siguiente:

$$H'(C_2, C_3, C_4 \dots C_{k+1}) = (H'(C_1, C_2, C_3 \dots C_k) - C_1 * b^k + C_k + 1) * b$$

Se usa como última operación la multiplicación, se garantiza que el número completo sea diferente, y no solo sus valores más pequeños.

2.2.5.2 Algoritmo todos contra todos

El algoritmo de Manber, todos contra todos[9], es el primer algoritmo de huella dactilar para detectar coincidencias entre múltiples documentos (Manber, 1994). Se llama todos contra todos dado que extrae los hashes de cada archivo para ser comparados con los demás, buscando coincidencias, sin excluir ningún valor.

Este algoritmo si bien fue un desarrollo importante, se considera ineficiente en la medida en que el extraer, mantener y comparar todos los hashes de los K-gram en los documentos implica un costo computacional muy alto, pues puede llegar a ser de mayor tamaño el índice de hashes al tamaño del documento inicial.

Texto dentro de uno de los documentos
(a) Texto inicial

Textodentrodeunodelosdocumentos
(b) Texto sin caracteres especiales

Texto extod xtode toden odent dentr entro ntrod trode rodeu
oden deuno eunod unode nodel odelo delos elosd losdo osdoc
sdocu docum ocume cumen ument mento entos
(c) 5-gram derivados del texto

20 41 17 14 76 25 89 39 01 23 30 46 28 14 29 43 71 26
94 73 92 41 05 38 91 28 63
(d) Hashes hipotéticos de los 5-gram

14 25 01 23 14 26 41 05 28
(e) Fingerprints en una ventana de tamaño 4

En un texto de 31 caracteres, con un k de 5, se obtienen 27 hashes. El número de comparaciones que se deben realizar no es significativamente menor. Una solución sería utilizar sólo algunos hashes generados, disminuyendo notablemente el número de comparaciones a realizar. Para este objetivo, una primera aproximación sería utilizar el hash en la iésima posición del anterior. Sin embargo, esta estrategia

[9] All-to-all matching en inglés.

incumpliría la tercera propiedad que debe tener un algoritmo de detección de similitudes, la independencia de posición. Modificar mínimamente la posición de un fragmento en el código podría variar el conjunto de hashes seleccionados. Por ejemplo, añadir 5 caracteres en el documento volcaría todos los K-grams en 5 posiciones, cambiando notablemente el resultado.

Otra estrategia adoptada por Manber, es la selección de los hashes que sean *0 mod p*[10]. Esta cumple correctamente con la tercera propiedad para algoritmos de detección de similitudes, pero pierde las garantías de encontrarlas de forma exitosa. Para un *p* cualquiera, existe un *g* que es la brecha entre la selección de dos hashes. *g* puede ser lo suficientemente grande para evadir hashes que son semejantes en dos documentos, dado que estos no son *0 mod p*. Esta selección es específica y dependiente del valor de *p*, permitiendo que el algoritmo garantice dichas similitudes sólo en términos de *p*.

[10] Significa que, al número obtenido por el hash, se le aplica la función módulo en base p. Si el resultado da 0, el hash cumple con la condición.

3 DISEÑO Y ESPECIFICACIONES

3.1 DEFINICIÓN DEL PROBLEMA

El plagio es una realidad que debe ser enfrentada por las instituciones educativas, ya que representan un hecho que atenta los deberes que posee un estudiante. Para enfrentar esta situación existen múltiples mecanismos que abarcan los casos de fraude más generales, los textos escritos. Sin embargo, para aquellos trabajos que son de programación, para detectar y corroborar casos de copia no se poseen procedimientos que tengan un costo asociado bajo y que presenten resultados sencillos de interpretar. De igual forma, de ser usados los mismos instrumentos que poseen los textos escritos, la sintaxis de lenguaje sería responsable del mayor porcentaje de ocurrencia de código repetido.

Poseer tales mecanismos para detectar el plagio en el desarrollo de software es vital por el impacto de su servicio y la capacidad de realizar este trabajo de forma automática, sin intervención de un usuario. Así, se omitiría el factor humano al momento de realizar la comparación, permitiendo a este ser el ente que da inicio al proceso y ser el que recibe el reporte final, para su entendimiento y validación. Adicionalmente, se tendrá acceso a información que hasta hoy es un trabajo complejo de generar. Por ejemplo, entre diferentes secciones de la misma clase, validar que dos o más archivos son similares, es una labor que requiere que los responsables de calificar permitan ver los archivos de todos los estudiantes y hacer la comparación manualmente.

Actualmente, existen herramientas para implementar estas funcionalidades. Dentro de las más reconocidas se encuentran MOSS, JPLAG y Marble. El software MOSS, creado por el profesor Alex Aiken en la universidad de Stanford, es el primer servicio que inició en la web, siendo una referencia a nivel mundial. Este, permite comparar hasta 250 archivos en 25 lenguajes de programación (Rademaker, 2010). Aunque no tiene licencia como software libre, su uso dentro del ambiente académico es gratuito y ofrecido desde un servidor de Stanford.

Teniendo en cuenta estas consideraciones y necesidades, en este documento se presentará la base para un proyecto que permita detectar la similitud en software. Esta base, tiene como objetivo obtener los fingerprints de los documentos que permitan la comparación eficiente entre archivos y posea la información de la posición de cada fingerprint dentro del archivo original. Para lograrlo, es necesario definir los requerimientos que debe cumplir el algoritmo.

3.2 ESPECIFICACIONES

3.2.1 REQUERIMIENTOS FUNCIONALES

Teniendo en cuenta que el algoritmo solo sigue una línea de proceso, los requerimientos funcionales se listan en orden y cada uno es prerrequisito del siguiente. Este proceso debe ser ejecutado cuando se tenga en disposición un nuevo archivo. La comparación correrá cuando el cliente decida realizarla, seleccionando qué archivos desea comparar.

Tabla 4. Formato de requerimientos funcionales

<id>999	*<nombre descriptivo>*
Descripción	El sistema deberá almacenar la información correspondiente a *<concepto relevante>.* En concreto*:*
Precondición	*<requisitos generales de los que depende>*
Postcondición	*<datos específicos sobre el concepto de salida>*
Importancia	*<importancia del requisito para el cliente>*
Prioridad	*<prioridad del requisito para la dirección del proyecto>*
Comentarios	*<comentarios adicionales sobre el requisito de información>*

Fuente: Autor.

El anterior, es el formato de especificación de cada requerimiento funcional.

Tabla 5. Requerimiento funcional 1

01	Tokenización
Descripción	El sistema deberá extraer y transformar la información correspondiente a los archivos que se desean comparar.
Precondición	Los archivos se encuentran descomprimidos
Postcondición	Se almacena en un archivo de texto una cadena de caracteres con la información en forma de tokens.
Importancia	Media
Prioridad	Alta
Comentarios	La salida del sistema se dará en formato <<Token,fila$>>

Fuente: Autor

Tabla 6. Requerimiento funcional 2

02	Creación K-grams
Descripción	El sistema deberá separar los caracteres de los tokens de forma que genere K-grams
Precondición	Cadena de caracteres en formato <<Token,fila$>>
Postcondición	Lista con todos los K-grams.
Importancia	Media
Prioridad	Alta
Comentarios	Cada K-gram debe tener la información relevante sobre su posición dentro del archivo original.

Fuente: Autor

Tabla 7. Requerimiento funcional 3

03	Hash sobre K-gram
Descripción	El sistema deberá crear el hash para cada valor de K-gram.
Precondición	Lista con todos los K-grams.
Postcondición	Lista de todos los K-grams con su valor convertido en hash.
Importancia	Media
Prioridad	Alta
Comentarios	El valor del K-gram pasa de ser una subcadena de caracteres a un número en hexadecimal

Fuente: Autor

Tabla 8. Requerimiento funcional 4

04	Búsqueda de Fingerprints
Descripción	El sistema deberá encontrar los hashes que serán marcados como fingerprints
Precondición	Lista de todos los K-grams con su valor convertido en hash.
Postcondición	Lista de los K-grams marcados como fingerprints
Importancia	Media
Prioridad	Alta
Comentarios	De la lista de los K-gram, debe seleccionar por medio de una estrategia eficiente cuáles deben ser considerados como Fingerprints

Fuente: Autor

Tabla 9. Requerimiento funcional 5

05	Almacenamiento de Fingerprints en Base de datos
Descripción	El sistema deberá almacenar la información correspondiente a los fingerprints en la base de datos PostgreSQL
Precondición	Lista de los K-grams marcados como fingerprints
Postcondición	Los fingerprints se encuentran almacenados en base de datos.
Importancia	Media
Prioridad	Alta
Comentarios	Se almacenan para su futura comparación si el cliente lo desea.

Fuente: Autor

3.2.2 ATRIBUTOS DE CALIDAD

Los requerimientos no funcionales que se desean tener en cuenta son aquellos relacionados con el tiempo de procesamiento del proyecto.

Tabla 10. Formato atributos de calidad

Escenario	***Escenario X***
Identificador	*idX*
Prioridad	*<Prioridad para el proyecto>*
Atributo de calidad	*<Atributo que impacta>*
Estímulo	*<Por qué se presenta>*
Ambiente	*<Ambiente de desarrollo del escenario>*
Medida esperada	*<Tiempo de respuesta, calidad de la respuesta>*

Fuente: Autor

El anterior es el formato para la presentación de atributos de calidad.

Tabla 11. Formato atributos de calidad 1

Escenario	**Escenario 1**
Identificador	Id1
Prioridad	Alta
Atributo de calidad	Desempeño
Estímulo	Generar Fingerprints con la información de los archivos referenciados.
Ambiente	Normal
Medida esperada	Se espera que la velocidad de respuesta del sistema sea inferior a 30 ms

Fuente: Autor

Se omiten posibles requerimientos no funcionales a razón que no se relacionan directamente con la línea de proceso descrita anteriormente.

3.2.3 SOLUCIÓN APROXIMADA

Una solución aproximada al problema sería aquella que tiene una cobertura en múltiples lenguajes de programación, incluyendo Java y que no cumple los requerimientos no funcionales frente archivos que presentan un código extenso, superando las 700 líneas de código.

Una solución es aceptable si se encuentra funcionalmente correcta y cumple sus requerimientos en ambientes de producción normales. Asimismo, es aceptable si su capacidad de entendimiento frente a los lenguajes de programación posee baja cobertura, pero se encuentra como una aplicación escalable y que responda correctamente frente al lenguaje Java.

3.3 RESTRICCIONES

Para el desarrollo de la solución, se establece una restricción a partir de consideraciones en términos de mantenimiento, desempeño y disponibilidad de la solución. Esta restricción, indica que solo es posible realizar la implementación en el lenguaje Elixir, permitiendo fragmentos de código en otros lenguajes si no es posible anexar esas funcionalidades dentro de Elixir. Este lenguaje es basado en Erlang, un lenguaje que soporta eficientemente la concurrencia, la distribución, la tolerancia y recuperación frente a fallos (Halmagean, 2018).

4. DESARROLLO DEL DISEÑO

Para el proceso de diseño, se realizó la búsqueda de los factores que permitieran dar como resultado la solución esperada. Se encontraron varias rutas alternas, las cuales se fueron descartando al considerar los requerimientos funcionales, los requerimientos no funcionales y las restricciones que debe tener el proyecto. Finalmente, luego de separar el proceso en tres componentes principales, se presenta el diseño de componentes:

Figura 19. Diseño de componentes

Fuente: Autor

En este, las interacciones se inician desde el archivo, el cual es consumido por un script de JS que es ejecutado por el programa principal para el preprocesamiento de los datos, logrando la tokenización. Luego, al llegar la información transformada, Elixir la procesa para su posterior almacenamiento y comparación en PostgreSQL

Figura 20. Esquema de componentes

Fuente: Autor

4.1 RECOLECCIÓN DE INFORMACIÓN.

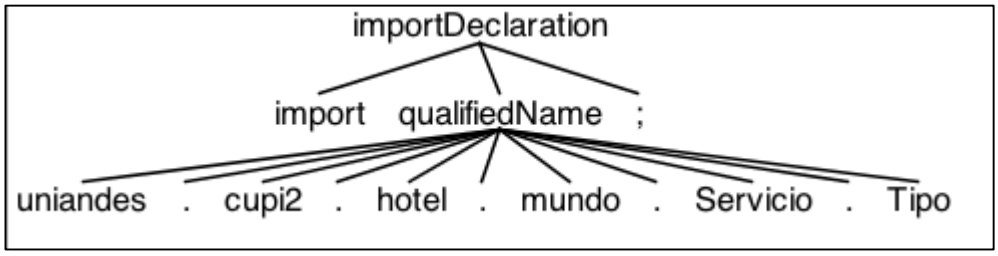
En la fase de diseño se encontraron tres partes de la solución que debían ser analizadas individualmente. La primera, es el proceso de transformación del lenguaje, seguido por la creación y selección de los hashes como fingerprints, para finalmente almacenar estos en una base de datos que permitiera realizar comparaciones y consultas sobre estas.

4.1.1 TRANSFORMACIÓN DEL LENGUAJE

Los lenguajes de programación también son conocidos como lenguajes formales, los cuales poseen la particularidad de poder ser definidos dentro de un árbol de

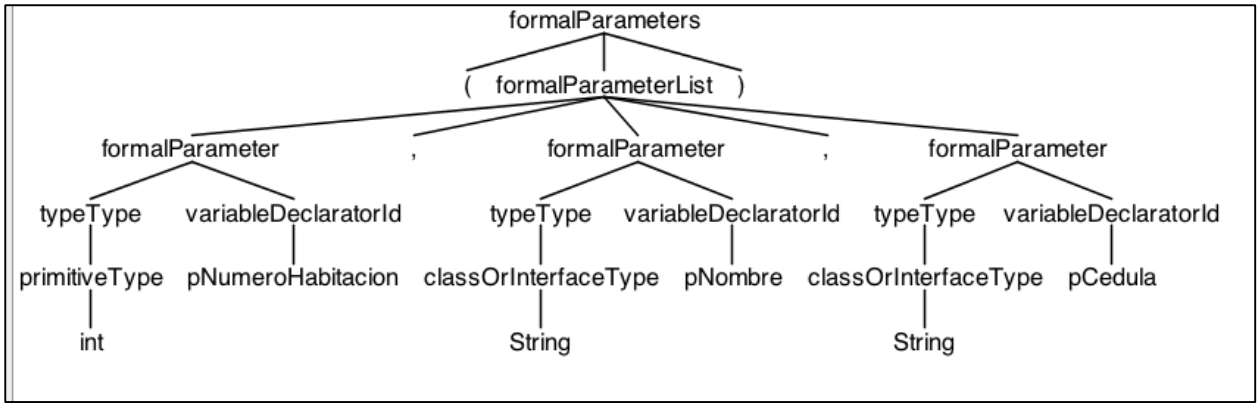
sintaxis abstracta, AST[11]. Este árbol es una forma de organizar el código fuente por bloques, donde cada bloque hace parte de un nivel de granularidad.

Figura 21. Ejemplo 1 de AST



Fuente: Autor. ANTLR4

Figura 22. Ejemplo 2 de AST



Fuente: Autor. ANTLR4

En los anteriores AST, se puede apreciar cómo se organiza cada palabra o símbolo como único elemento de los bloques con la mayor granularidad. Estos, son los que se reconocen como Tokens, ya que identifican cada uno de los elementos dentro de una familia. De esta forma, es posible transformar el código fuente, permitiendo eliminar los caracteres y valores que se desean despreciar e indicando el lugar en el que se encuentra cada token.

Para lograr este objetivo, se encuentran disponibles diversas herramientas que generan el árbol, pero ninguna cumple la restricción del lenguaje de programación. Entre las herramientas encontradas, están ANTLR4, APG, Bison, CookCC y jacc. Finalmente, se sugiere seguir el esquema de ANTLR4, pues integra una gramática

[11] Abstract Syntax Tree

reducida de Java, permitiendo generar un AST con un alto desempeño, sin importar el tamaño de la entrada. Las otras gramáticas de Java disponibles en esta plataforma y en las demás, no presentaban respuesta luego de un par de minutos, siendo casos de suma ineficiencia para el proyecto.

4.1.2 WINNOWING

Para encontrar los fingerprints de los documentos a partir de sus tokens, es necesario crear un esquema de K-grams buscando soluciones que tienen en cuenta el desempeño. Se encontró que el algoritmo de Winnowing garantiza la aparición de fingerprints con un espacio máximo de W entre un fingerprint y el otro, maximizando la eficiencia y cumpliendo correctamente las tres características deseadas en un algoritmo de este tipo.
Winnowing funciona a base de K-grams, los cuales son cadenas de caracteres de tamaño K, basados en los tokens. Un token puede pertenecer a uno o más K-grams, dependiendo del tamaño que este posea y qué tan grande o pequeño sea K.

A partir de estos K-grams, el valor es reemplazado por un número en hexadecimal por medio de una función de hash, al igual que en algoritmos anteriormente desarrollados, como todos contra todos o karp-robin. El cambio inicia en la selección de fingerprints, pues la estrategia que ha caracterizado a Winnowing lo ha convertido en un algoritmo local de búsqueda de fingerprints. Esta selección se hace por medio de una ventana de tamaño W donde W indica la cantidad de K-grams que se desean valorar en un mismo momento.

Al tener una ventana con W K-grams al tiempo, se decide seleccionar uno de esta ventana como fingerprint, para luego mover la ventana una posición a la derecha. Para tener control de la selección y que no sea aleatoria, se impone la condición de seleccionar siempre al menor de los K-grams dentro de la ventana. Si existe más de uno, se debe seleccionar el que se encuentra más a la derecha. Las variables K y W han sido diseñadas para que el implementador decida el tamaño óptimo para la situación.

20 41 17 14 76 25 89 39 01 23 30 46
(a) Hipotéticos hashes de K-grams

(20,41,17,**15**,76) (41,17,15,76,**14**)
(17,15,76,14,89)
(15,76,14,89,39) (76,14,89,39,**01**)
(14,89,39,01,23) (89,39,01,23,30) (39,01,23,30,46)
(b) Ventana de tamaño 5

15, 14, 01
(c) Fingerprints seleccionados por Winnowing

La noción de escoger el hash con el menor valor permite reducir la cantidad de fingerprints seleccionados, pues en ventanas que se sobreponen entre sí en uno o más de sus elementos, pueden estar seleccionando el mismo valor. Adicionalmente, se sigue garantizando que la distancia entre los fingerprints es tan grande como W lo sea (Schleimer, Wilkerson, Aiken & Berkeley, (n.d)).

Como se había mencionado, Winnowing tiene un componente importante en términos de desempeño. Lo que hace que sea eficiente comparado con otros algoritmos igual de precisos, es que el tamaño de la salida es una fracción de la entrada. Se conoce como densidad a la proporción de fingerprints seleccionados sobre el tamaño total de hashes posibles. La densidad de Winnowing es de (2/w+1), siendo alrededor del 33% más alto que el límite inferior de los algoritmos locales basados en fingerprints (1,5/w+1), pero menor a las densidades reportadas en algoritmos basados en 0 mod p, que sugieren una densidad relativa a ((1 + Ln w) / w) (Schleimer, Wilkerson, Aiken & Berkeley, (n.d)).

4.1.3 POSTGRESQL

Para almacenar los fingerprints de los documentos, se observaron diferentes bases de datos no relacionales, dada la importancia de la consulta sobre la escritura. Siguiendo los requerimientos y las restricciones, se encuentra que PostreSQL posee una integración ligera y nativa desde el lenguaje de programación Elixir. Por este motivo, se decide utilizar la base de datos no relacional PostgreSQL.

5. IMPLEMENTACIÓN

A partir del diseño, se decide realizar la implementación de las tres partes, enunciadas a partir de ahora como dos, pues Elixir se encarga nativamente de manejar la base de datos. La implementación abarca desde el momento en que se leen los archivos que se desean comparar, hasta el almacenamiento de los fingerprints en la base de datos.

Para el manejo del AST, se consultaron diferentes referencias y opciones para desarrollarlo en Elixir. A partir de los parsers[12] existentes, se encuentra que se encuentra fuera del alcance del proyecto generar el parser en Elixir, principalmente por la creación del árbol identificando Tokens, a partir de una gramática que el lenguaje también debe tener la capacidad de entender. Por lo tanto, se decide implementar un fragmento de código en uno de los lenguajes soportado directamente por ANTLR4, el cual al calcular el costo de llamarlo desde Elixir se escogió el fragmento de JavaScript.

Dentro del observable que genera ANTLR4 para atravesar el árbol, se genera el desarrollo del código que permite filtrar del mismo los caracteres que no se desean tener en cuenta, reemplazando palabras no reservadas del lenguaje por un valor genérico.

Figura 23. Validación de Tokens

```
//validacion de si es un valor normal de Java y no algo específico del ejercicio
validacion = (tmp)=>{
    //Verifica que no es un carácter especial
    if(caracteres_especiales.filter(d => d === tmp).length !== 0){
        return undefined;
    }
    //Verifica que haga parte de la JSL o de sus palabras reservadas
    if(java_standart_library.filter(d=> d === tmp).length !== 0){
        return tmp;
    }
    //en caso contrario
    return "var";
}
```

Fuente: Autor. NOTEPAD++

Esta validación es escalable a múltiples lenguajes de programación, configurando ANTLR4 para las diferentes gramáticas, listando las palabras reservadas y similares a la JSL. También, es ejecutada por la función que se encuentra dentro de los observables de cada Token.

[12] Software que permite analizar sintácticamente una entrada, teniendo en cuenta una gramática específica. Este análisis se realiza mediante la creación de un árbol de sintaxis abstracto.

Figura 24. Cargue de Tockens

```
cargue = (ctx, texto) =>{
    ctx.children.map((d)=>{
        if(d.children === undefined){
            var tmp = validacion(d.getText());
            if(tmp != undefined){
                texto(tmp + "," + ctx.start.line);
            }
        }
    });
}
```

Fuente: Autor. NOTEPAD++

Dada la implementación del árbol de ANTLR4 en JavaScript, sólo si el hijo de mi hijo es inexistente, significa que mi hijo es una hoja. Si el valor de la validación es diferente a indefinido, es un Token válido y debe ser enviado en la cadena de caracteres de salida con su respectiva ubicación en el archivo original.

Elixir realiza un llamado a este script cuando reconoce todos los archivos que debe analizar en ese momento, recibiendo una cadena de caracteres por cada uno de los archivos analizados por el parser. A partir de este punto, entran a jugar las variables K y W enunciadas anteriormente, ya que del valor de estas dependerá la cantidad de falsos positivos o falsos negativos que encuentre el usuario al realizar la comparación.

Por ejemplo, en Java es usual encontrar líneas de código similares a esta:

Figura 25. Ejemplo 2 de constante en Java

```
private final static String MI_CONSTANTE = "mi constante";
```

Fuente: Autor

Donde se está declarando y asignando una constante. Si contamos los caracteres por palabra y el espacio después de este, tendríamos 58 caracteres.

Tabla 12. Descomposición de constante en caracteres

private	final	static	String	MI_CONSTANTE	=	“mi	constante”;
8	6	7	7	13	2	4	11

Fuente: Autor

Es decir, un K cercano a 58 caracteres en Java puede dar un buen acercamiento para encontrar plagio, superando la barrera de la sintaxis. Por otro lado, también se evidencia que parte de estos caracteres han de perderse dentro del procesamiento

que realiza el parser, bajando la cantidad de caracteres finales a 40. Es decir, un K en el rango entre 40 y 58 para Java sería un K para experimentar.

En el caso de Python, las instrucciones tienden a ser de un tamaño menor, incluso más corto que la mitad de los caracteres que posee Java para una sola instrucción. Por esta razón, es imprescindible tener un valor de K diferente para cada lenguaje de programación.

Así mismo, el tamaño de W es a considerar y que, lo único que se reconoce de él, es la métrica que permite estimar la cantidad de fingerprints que el algoritmo va a generar. En otras palabras, W tiene un tamaño dependiente de la capacidad de la base de datos más la granularidad que desea se tenga el programa. Entre más pequeño W, más fingerprints va a generar Winnowing, con una distancia máxima de W.

Luego de definir un aproximado de las variables W y K, se realiza la transformación de los tokens a caracteres individuales, cada uno conociendo su fila dentro del archivo original. Esta implementación usa recursividad de tipo tail recursion, permitiendo que se ejecute sin número de veces sin que el programa detenga su ejecución por superar los límites de memoria o de pila.

Figura 26. Implementación de la transformación de Tokens a caracteres

```
def startProcess([head|tail], saved) do
  [token, pFila] = String.split(head,",")
  {fila,_} = Integer.parse(pFila)
  lista_tmp_char = String.split(token, "", trim: true)
  lista_char_reverse = Enum.map(lista_tmp_char, fn char ->
    [char|[fila|[]]]
  end)
  lista_char = reverse(lista_char_reverse, saved)
  startProcess(tail, lista_char)
end
```

Fuente: Autor. NOTEPAD++

El resultado de este subproceso se presenta de la siguiente forma:
[[charN,FilaN],[CharN-1,FilaN],.....,[Char0,Fila0]]

Luego, se crean los K-grams y al mismo tiempo se reemplaza la cadena de caracteres por su hexadecimal, omitiendo aquellos que se encuentren dentro de un archivo de esqueleto, para finalmente pasar por un subproceso que selecciona los primeros W K-grams, seleccionado el de menor valor, expulsando el K-gram que se encuentra en la primera posición, añadiendo el siguiente y repitiendo el proceso desde la selección.

Al devolver la lista de estos fingerprints, Elixir los guarda dentro de la base de datos, donde el registro tiene los atributos que definen el documento específico. Este registro queda listo para ser comparado con los demás documentos que han sido almacenados.

Las tablas en las que se guardan estos fingerprints, se diferencian dependiendo de su uso. La primera, se refiere a los archivos esqueletos, llamada winesqueletos, en la cual se almacenan los fingerprints de los archivos entregados por la institución académica a los estudiantes, los cuales deben ser ignorados para evitar falsos positivos. La segunda tabla, es en la que se almacenan los archivos de los estudiantes. Esta tabla se encuentra como winarchivoscomparaciones, donde se almacenan los fingerprints seleccionados para cada documento.

Cada registro cuenta con varios datos adicionales que hacen referencia a los casos de uso del proyecto web, guardando información del estudiante, de la ubicación del archivo, fecha de cargue, orden de fingerprints para cada documento, con su respectivo fingerprint y filas de inicio y fin.

Tabla 13. Estructura básica Winesqueletos

id	id_fingerprint	fingerprint	nombre	ubicacion	fila_inicial	fila_final	actividad_id	inserted_at	updated_at
x	01	00DE...	estudia	FileName	01	05	y	date()	date()

Fuente: Autor. Estructura básica de la tabla winesqueletos

Tabla 14. Estructura básica Winarchivoscomparaciones

id	id_fingerprint	fingerprint	nombre	ubicacion	fila_inicial	fila_final	actividad_id	estudiante_id	inserted_at	updated_at
x	01	00DE...	estudia	FileName	01	05	y	z	date()	date()

Fuente: Autor. Estructura básica de la tabla winarchivoscomparaciones

6. RESULTADOS

A partir de la implementación generada, se toman resultados a partir del cargue de algunos documentos en Java. Estos, se presentan a continuación como screenshots tomados directamente de la base de datos después de correr el algoritmo. Este corre con tamaños de W y K iguales a 150.

Figura 27. Datos almacenados en Winesqueletos

```
desisoft_dev=# select * from winesqueletos;
  id   | id_fingerprint |          fingerprint           |              nombre              |            ubicacion             | fila_inicial | fila_final | actividad_id
|      inserted_at       |         updated_at
-------+----------------+--------------------------------+----------------------------------+----------------------------------+--------------+------------+--------------
+------------------------+------------------------
 23455 |              1 | 03665E9CA36C75658A692D7E07A948FD | ISIS1206_Banda.java              | ISIS1206_Banda.java              |           99 |        135 |            2
| 2018-12-17 21:44:43.825488 | 2018-12-17 21:44:43.825504
 23456 |              2 | 03D16C5028E57BC56CD092ECF9A840CD | ISIS1206_Banda.java              | ISIS1206_Banda.java              |           79 |        106 |            2
| 2018-12-17 21:44:43.835258 | 2018-12-17 21:44:43.83527
 23457 |              3 | 011309C23200E6A758715B472D894FE8 | ISIS1206_Banda.java              | ISIS1206_Banda.java              |           66 |         90 |            2
| 2018-12-17 21:44:43.83915  | 2018-12-17 21:44:43.839159
 23458 |              4 | 014C1639B6D94835F4ACB1DCACB632FC | ISIS1206_Banda.java              | ISIS1206_Banda.java              |           62 |         66 |            2
| 2018-12-17 21:44:43.842788 | 2018-12-17 21:44:43.842798
 23459 |              5 | 0125E0A1A6BA45FFEB782F1EB67B47E2 | ISIS1206_Banda.java              | ISIS1206_Banda.java              |           42 |         65 |            2
| 2018-12-17 21:44:43.846517 | 2018-12-17 21:44:43.846526
 23460 |              1 | 003C97ACC01D721D0A751CFB2AF39742 | ISIS1206_BandaTest.java          | ISIS1206_BandaTest.java          |           66 |         70 |            2
| 2018-12-17 21:44:44.4068   | 2018-12-17 21:44:44.406809
 23461 |              2 | 007420E879A486F26755409DA23B1134 | ISIS1206_BandaTest.java          | ISIS1206_BandaTest.java          |           66 |         69 |            2
| 2018-12-17 21:44:44.412069 | 2018-12-17 21:44:44.412079
 23462 |              3 | 03D309F50A3C6ADE3B91416D1BF87835 | ISIS1206_BandaTest.java          | ISIS1206_BandaTest.java          |           42 |         68 |            2
| 2018-12-17 21:44:44.415556 | 2018-12-17 21:44:44.415565
 23463 |              4 | 00789B8DB6BF5955A2B7CBA2C3649588 | ISIS1206_BandaTest.java          | ISIS1206_BandaTest.java          |           17 |         44 |            2
| 2018-12-17 21:44:44.419389 | 2018-12-17 21:44:44.419336
 23464 |              1 | 00B1DF424479A4EFD98312DC9F306E5C | ISIS1206_DialogoCrearBanda.java  | ISIS1206_DialogoCrearBanda.java  |          206 |        214 |            2
| 2018-12-17 21:44:45.548688 | 2018-12-17 21:44:45.548697
 23465 |              2 | 02C249A7704B6759B93E10B553A3CFD8 | ISIS1206_DialogoCrearBanda.java  | ISIS1206_DialogoCrearBanda.java  |          200 |        206 |            2
| 2018-12-17 21:44:45.554792 | 2018-12-17 21:44:45.554804
 23466 |              3 | 0222D26A52EBF88CD359342316A51F8A | ISIS1206_DialogoCrearBanda.java  | ISIS1206_DialogoCrearBanda.java  |          187 |        204 |            2
| 2018-12-17 21:44:45.557148 | 2018-12-17 21:44:45.557158
 23467 |              4 | 000A6B445DA1276080221 0A0C485B7A5 | ISIS1206_DialogoCrearBanda.java  | ISIS1206_DialogoCrearBanda.java  |          185 |        202 |            2
| 2018-12-17 21:44:45.559798 | 2018-12-17 21:44:45.55981
 23468 |              5 | 00DE110F2758B1F5A1BEE1630F5884D0 | ISIS1206_DialogoCrearBanda.java  | ISIS1206_DialogoCrearBanda.java  |          182 |        200 |            2
| 2018-12-17 21:44:45.562231 | 2018-12-17 21:44:45.562242
 23469 |              6 | 0111B11E48D4BE3570613D004EE3ED16 | ISIS1206_DialogoCrearBanda.java  | ISIS1206_DialogoCrearBanda.java  |          178 |        186 |            2
| 2018-12-17 21:44:45.564602 | 2018-12-17 21:44:45.564612
 23470 |              7 | 0172A8776273CE5C408F3B19CECFAB73 | ISIS1206_DialogoCrearBanda.java  | ISIS1206_DialogoCrearBanda.java  |          171 |        180 |            2
| 2018-12-17 21:44:45.566813 | 2018-12-17 21:44:45.566822
 23471 |              8 | 019470192AA041770685A71B56C04F3D | ISIS1206_DialogoCrearBanda.java  | ISIS1206_DialogoCrearBanda.java  |          168 |        176 |            2
```

Fuente: Autor. TERMINAL

Fingerprints de la tabla winesqueletos. Estos son los fingerprints que pueden ser comunes a todos los documentos de estudiantes y que no deben ser considerados para casos de plagio. Como segunda fila de cada registro, aparecen los valores de las últimas dos columnas.

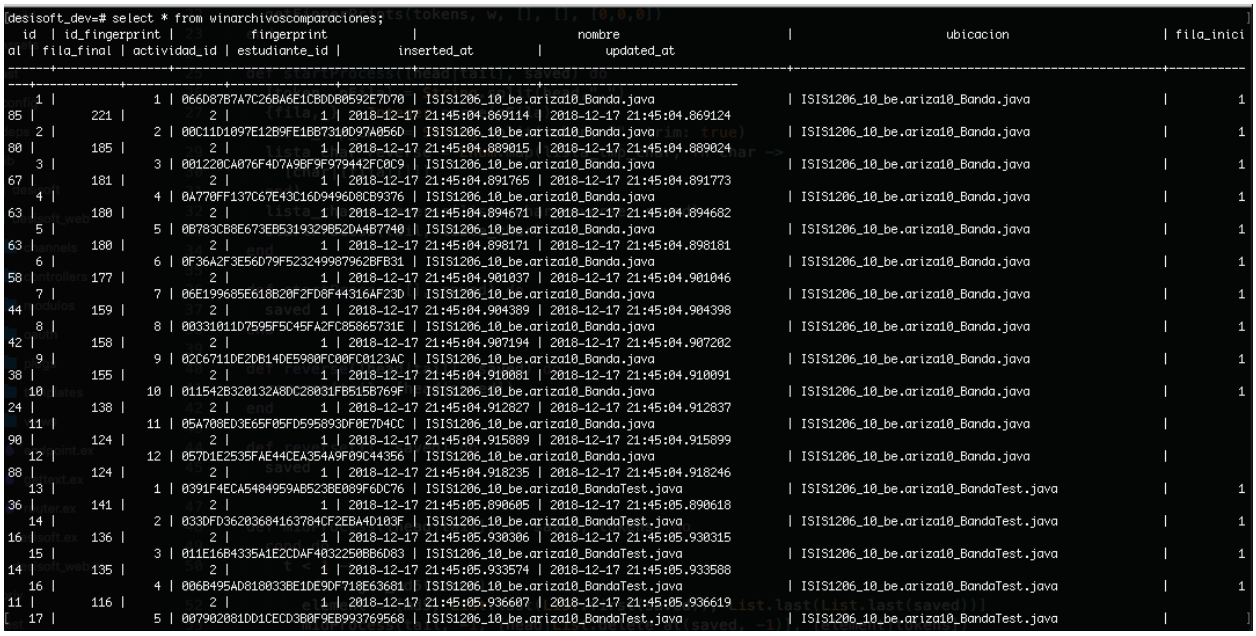
Figura 28. Filtro de datos en Winesqueletos

```
desisoft_dev=# select * from winesqueletos where fingerprint = '03665E9CA36C75658A692D7E07A948FD';
  id   | id_fingerprint |          fingerprint           |       nombre       |      ubicacion     | fila_inicial | fila_final | actividad_id |       inserted_at       |
    updated_at
-------+----------------+--------------------------------+--------------------+--------------------+--------------+------------+--------------+-------------------------+--
-------------------------
 23455 |              1 | 03665E9CA36C75658A692D7E07A948FD | ISIS1206_Banda.java | ISIS1206_Banda.java |           99 |        135 |            2 | 2018-12-17 21:44:43.825488 | 2
018-12-17 21:44:43.825504
(1 row)

(END)
```

Fuente: Autor. TERMINAL

Al buscar un fingerprint en específico, este se encuentra una sola vez, validando que no existen subcadenas de caracteres entre todos los archivos esqueleto que sean iguales a la reportada con este fingerprint, que se marca entre las líneas 99 y 135 del archivo *_Banda.java

Figura 29. Datos almacenados en Winarchivoscomparaciones



```
desisoft_dev=# select * from winarchivoscomparaciones;
  id  | id_fingerprint |              fingerprint             |                 nombre                 |                 ubicacion                 | fila_inici
al | fila_final | actividad_id | estudiante_id |       inserted_at       |       updated_at
------+----------------+--------------------------------------+----------------------------------------+-------------------------------------------+-----------
---+------------+--------------+---------------+-------------------------+-------------------------
    1 |              1 | 066D87B7A7C26BA6E1CBDDB0592E7D70 | ISIS1206_10_be.ariza10_Banda.java      | ISIS1206_10_be.ariza10_Banda.java         |         1
85 |        221 |            2 |             1 | 2018-12-17 21:45:04.869114 | 2018-12-17 21:45:04.869124
    2 |              2 | 00C11D1097E12B9FE1BB7310D97A056D | ISIS1206_10_be.ariza10_Banda.java      | ISIS1206_10_be.ariza10_Banda.java         |         1
80 |        185 |            2 |             1 | 2018-12-17 21:45:04.889015 | 2018-12-17 21:45:04.889024
    3 |              3 | 001220CA076F4D7A9BF9F979442FC0C9 | ISIS1206_10_be.ariza10_Banda.java      | ISIS1206_10_be.ariza10_Banda.java         |         1
67 |        181 |            2 |             1 | 2018-12-17 21:45:04.891765 | 2018-12-17 21:45:04.891773
    4 |              4 | 0A770FF137C67E43C16D9496D8CB9376 | ISIS1206_10_be.ariza10_Banda.java      | ISIS1206_10_be.ariza10_Banda.java         |         1
63 |        180 |            2 |             1 | 2018-12-17 21:45:04.894671 | 2018-12-17 21:45:04.894682
    5 |              5 | 0B783CB8E673EB5319329B52DA4B7740 | ISIS1206_10_be.ariza10_Banda.java      | ISIS1206_10_be.ariza10_Banda.java         |         1
63 |        180 |            2 |             1 | 2018-12-17 21:45:04.898171 | 2018-12-17 21:45:04.898181
    6 |              6 | 0F36A2F3E56D79F523249987962BFB31 | ISIS1206_10_be.ariza10_Banda.java      | ISIS1206_10_be.ariza10_Banda.java         |         1
58 |        177 |            2 |             1 | 2018-12-17 21:45:04.901037 | 2018-12-17 21:45:04.901046
    7 |              7 | 06E199685E618B20F2FD8F44316AF23D | ISIS1206_10_be.ariza10_Banda.java      | ISIS1206_10_be.ariza10_Banda.java         |         1
44 |        159 |            2 |             1 | 2018-12-17 21:45:04.904389 | 2018-12-17 21:45:04.904398
    8 |              8 | 00331011D7595F5C45FA2FC85865731E | ISIS1206_10_be.ariza10_Banda.java      | ISIS1206_10_be.ariza10_Banda.java         |         1
42 |        158 |            2 |             1 | 2018-12-17 21:45:04.907194 | 2018-12-17 21:45:04.907202
    9 |              9 | 02C6711DE2DB14DE5980FC00FC0123AC | ISIS1206_10_be.ariza10_Banda.java      | ISIS1206_10_be.ariza10_Banda.java         |         1
38 |        155 |            2 |             1 | 2018-12-17 21:45:04.910081 | 2018-12-17 21:45:04.910091
   10 |             10 | 011542B320132A8DC28031FB515B769F | ISIS1206_10_be.ariza10_Banda.java      | ISIS1206_10_be.ariza10_Banda.java         |         1
24 |        138 |            2 |             1 | 2018-12-17 21:45:04.912827 | 2018-12-17 21:45:04.912837
   11 |             11 | 05A708ED3E65F05FD595893DF0E7D4CC | ISIS1206_10_be.ariza10_Banda.java      | ISIS1206_10_be.ariza10_Banda.java         |
90 |        124 |            2 |             1 | 2018-12-17 21:45:04.915889 | 2018-12-17 21:45:04.915899
   12 |             12 | 057D1E2535FAE44CEA354A9F09C44356 | ISIS1206_10_be.ariza10_Banda.java      | ISIS1206_10_be.ariza10_Banda.java         |
88 |        124 |            2 |             1 | 2018-12-17 21:45:04.918235 | 2018-12-17 21:45:04.918246
   13 |              1 | 0391F4ECA5484959AB523BE089F6DC76 | ISIS1206_10_be.ariza10_BandaTest.java  | ISIS1206_10_be.ariza10_BandaTest.java     |         1
36 |        141 |            2 |             1 | 2018-12-17 21:45:05.890605 | 2018-12-17 21:45:05.890618
   14 |              2 | 033DFD362036841637B4CF2EBA4D103F | ISIS1206_10_be.ariza10_BandaTest.java  | ISIS1206_10_be.ariza10_BandaTest.java     |         1
16 |        136 |            2 |             1 | 2018-12-17 21:45:05.930306 | 2018-12-17 21:45:05.930315
   15 |              3 | 011E16B4335A1E2CDAF4032250BB6D83 | ISIS1206_10_be.ariza10_BandaTest.java  | ISIS1206_10_be.ariza10_BandaTest.java     |         1
14 |        135 |            2 |             1 | 2018-12-17 21:45:05.933574 | 2018-12-17 21:45:05.933588
   16 |              4 | 006B495AD818033BE1DE9DF718E63681 | ISIS1206_10_be.ariza10_BandaTest.java  | ISIS1206_10_be.ariza10_BandaTest.java     |         1
11 |        116 |            2 |             1 | 2018-12-17 21:45:05.936607 | 2018-12-17 21:45:05.936619
   17 |              5 | 007902081DD1CECD3B0F9EB993769568 | ISIS1206_10_be.ariza10_BandaTest.java  | ISIS1206_10_be.ariza10_BandaTest.java     |
```

Fuente: Autor. TERMINAL

Fingerprints de la tabla winarchivoscomparaciones. Estos son los fingerprints que identifican de forma única y precisa cada uno de los documentos de estudiantes y que no deben presentar relación entre ellos, en caso de ser originales. Como segunda fila de cada registro, aparecen los valores de las últimas seis columnas.

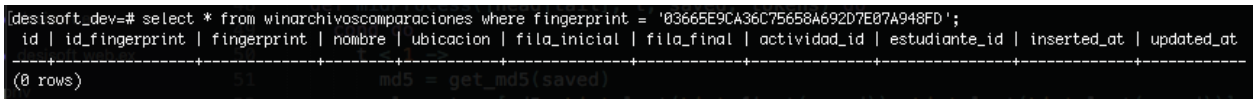
Figura 30. Filtro de datos en Winarchivoscomparaciones

```
desisoft_dev=# select * from winarchivoscomparaciones where fingerprint = '066D87B7A7C26BA6E1CBDDB0592E7D70';
 id | id_fingerprint |              fingerprint             |                 nombre                 |                 ubicacion                 | fila_inicial | fila_final | actividad_id | estud
iante_id |       inserted_at       |       updated_at
----+----------------+--------------------------------------+----------------------------------------+-------------------------------------------+--------------+------------+--------------+------
---------+-------------------------+-------------------------
  1 |              1 | 066D87B7A7C26BA6E1CBDDB0592E7D70 | ISIS1206_10_be.ariza10_Banda.java | ISIS1206_10_be.ariza10_Banda.java |          185 |        221 |            2 |
       1 | 2018-12-17 21:45:04.869114 | 2018-12-17 21:45:04.869124
(1 row)
```

Fuente: Autor. TERMINAL

Al buscar un fingerprint en específico, este se encuentra una sola vez, validando que no existen subcadenas de caracteres entre todos los archivos de estudiantes que sean iguales a la reportada con este fingerprint, que se marca entre las líneas 185 y 221 del archivo *_be.ariza10_Banda.java

Figura 31. Filtro de datos en Winarchivoscomparaciones 2



Fuente: Autor. TERMINAL

Al consultar un fingerprint que se encuentra dentro de la tabla winesqueletos en la tabla winarchivoscomparaciones, el resultado es sin coincidencias, verificando que los fingerprints de los esqueletos son omitidos para los archivos de estudiantes. Este fingerprint es el mismo con el que se realizó la consulta dentro de la tabla winesqueletos.

7. CONCLUSIONES

7.1 DISCUSIÓN

Se detalla una solución aproximada en el presente documento, desarrollada por medio de la implementación del algoritmo Winnowing. Esta solución se considera aproximada dado que cumple sus requerimientos en ambientes de producción normales. Asimismo, es aceptable su capacidad de entendimiento frente a los lenguajes de programación dado que es una aplicación escalable y que, actualmente, su uso estará enfocado para el lenguaje implementado.

El trabajo documentado se desarrolló en los tiempos especificados, contando también el tiempo de aprendizaje y búsqueda de las herramientas a utilizar para llevar a cabo la implementación. De igual forma, era deseable su entrega con la comparación en marcha, que de la forma en que se diseñó se realizaría desde la base de datos, creando una nueva tabla con los resultados de las comparaciones.

Gracias a la restricción tecnológica impuesta y al desarrollo realizado, el proyecto es capaz de sostenerse y mantener una alta disponibilidad y desempeño, pues la implementación no entorpece en gran magnitud la eficiencia natural de Elixir. Dado que se debió implementar una parte importante del desarrollo desde otro lenguaje, este puede presentar fallos en estos aspectos, disminuyendo estos atributos de calidad del proyecto en general. Estos casos se han de presentar al momento de cargar AST de códigos de más de 700 líneas de código aproximadamente, dependiendo de la capacidad del servidor.

Esta limitación se conoció en el momento en que se presentó el problema de encontrar o implementar un parser para el AST que correspondiera correctamente con Elixir. Por lo tanto, fue necesario evaluar la solución menos costosa y de rápida implementación para presentar los resultados. Es entonces que se comenta que esta limitación no permite escalar el proyecto a documentos con un extenso número de líneas de código, a menos que se desee sacrificar los atributos de calidad del algoritmo.

Los resultados presentados son satisfactorios para la continuación del proyecto, ya que a partir de estos es posible comparar documentos a partir de consultas SQL. Teniendo en cuenta esto, si el servidor se presenta en un hardware ideal para una base de datos, el desempeño de la aplicación tiene la posibilidad de mejorar.

Finalmente, dadas las especificaciones, falta la implementación de la última parte del algoritmo, encargada de la comparación y almacenamiento de los resultados, pues estos últimos son los que deben estar disponibles para ser leídos por el proyecto web para el consumo del usuario.

7.2 TRABAJO FUTURO

Como trabajo futuro, se encuentra de primera mano realizar el análisis para determinar los valores ideales de las constantes K y W dentro del algoritmo para cada lenguaje de programación que se desee implementar, incluyendo el que ya se encuentra implementado, Java.

Una segunda consideración debería ser completar la comparación, permitiendo conocer si dos o más fingerprints seguidos en dos documentos presentan relación, lo que conlleva a considerar que el texto entre estos también es sospechoso de ser similar en ambos archivos.

Como tercera consideración, se encuentra el transformar el procesamiento del árbol de sintaxis, o AST, al lenguaje Elixir, permitiendo explorar nuevas implementaciones y posibilitando la mejora de los atributos de calidad del proyecto.

8. REFERENCIAS

1. Armstrong, J. (2003). Making reliable distributed systems in the presence of so d ware errors, (November). Retrieved from http://erlang.org/download/armstrong_thesis_2003.pdf%0A

2. Bowyer, K. W., & Hall, L. O. (n.d.). Experience Using ” MOSS ” to Detect Cheating On Programming Assignments. Retrieved from https://www3.nd.edu/~kwb/nsf-ufe/1110.pdf%0A

3. Djuric, Z., & Gasevic, D. (n.d.). A Source Code Similarity System for Plagiarism Detection. Retrieved from https://pdfs.semanticscholar.org/ddac/34c94c550acc8cdbb4343f5ce0e95ff60359.pdf%0A

4. Gallo, D., Cernuda, A., Cueva, J., Díaz, M., García, M. & Redondo, J. (2003). Reflexiones y experiencias sobre la enseñanza de POO como único paradigma. Retrieved from http://digibuo.uniovi.es/dspace/handle/10651/30804

5. Halmagean, C. (2018). Why you too should learn Elixir. Retrieved from https://mixandgo.com/learn/why-you-too-should-learn-elixir

6. K-mer (n.d.) En Wikipedia. La enciclopedia libre. Recuperado el 20 de Agosto de 2018 de https://en.wikipedia.org/wiki/K-mer

7. Kölling, M. (1999). The problem of teaching object-oriented programming, 11(8), 8–15. Retrieved from https://core.ac.uk/download/pdf/62989.pdf

8. Lagos, P. S. (n.d.). Ingeniería de software educativo, teorías y metodologías que la sustentan, 1–9. Retrieved from http://inf.udec.cl/~revista/ediciones/edicion6/isetm.PDF

9. Lecture, A., & Karp-rabin, T. (2014). The Karp-Rabin Algorithm (a . k . a . the “Fingerprint” Method), 1976, 2–4. Retrieved from http://www.cs.cmu.edu/afs/cs/academic/class/15451-f14/www/lectures/lec6/karp-rabin-09-15-14.pdf%0A

10. Rabin, M. (1987). Efficient randomized, 31(2). Retrieved from https://pdfs.semanticscholar.org/c47d/151f09c567013761632c89e237431c6291a2.pdf%0A

11. Rademaker, P. (2010). A comparison of plagiarism detection tools, (June). Retrieved from

https://pdfs.semanticscholar.org/81de/497e2a2a58dfb1bd4ee9dced5f0c0e009e52.pdf%0A

12. RAE. 2018 *significado plagio.* Retrieved from http://dle.rae.es/?id=TIf06In

13. Ram, A. O. (2014). Python como primer lenguaje de programación, (May). Retrieved from https://www.researchgate.net/profile/Ariel_Ortiz/publication/228743292_Python_como_primer_lenguaje_de_programacion/links/00b4952125b19d49aa000000/Python-como-primer-lenguaje-de-programacion.pdf

14. Rodríguez, A. S., & Rica, U. D. C. (2012). El plagio y su impacto a nivel académico y profesional. Retrieved from http://eprints.rclis.org/19890/1/2-1-2.pdf

15. Schleimer, S., Wilkerson, D. S., Aiken, A., & Berkeley, U. C. (n.d.). Winnowing: Local Algorithms for Document Fingerprinting. Retrieved from http://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf%0A
16. Thesis, B., & Bostan, A. S. (2017). Winnowing Algorithm for Program Code, (July). Retrieved from https://www.sts.tuhh.de/pw-and-m-theses/2017/abdul17.pdf

17. Manber, U. (1994). Finding similar files in a large file system. In USENIX Winter 1994 Technical Conference, pp. 1–10.