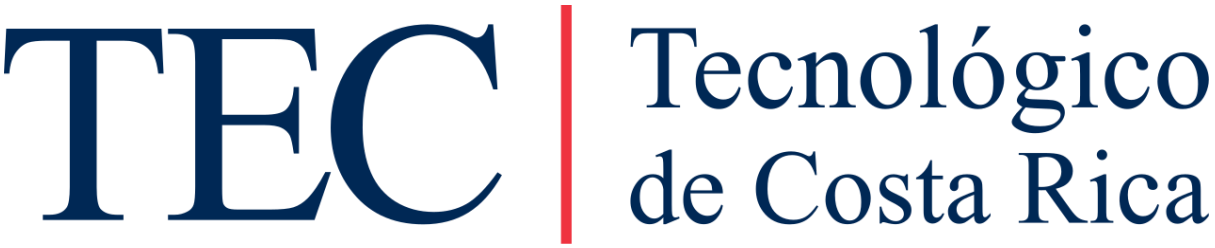
Instituto Tecnológico de Costa Rica



Proyecto #2
Investigación sobre PMD (Programming Mistake Detector)

Curso
Aseguramiento de la Calidad

Profesora
María Auxiliadora Mora Cross

Alumno
Jonathan Quesada Salas

Carnet
2020023583

Alajuela

Octubre 2020

ÍNDICE

Introducción **3**

Descripción de la herramienta **3**

- Características 3
- Requerimientos 3
- Ventajas 3
- Desventajas 3
- Otros detalles a resaltar para información del grupo 3

Metodología **3**

- Justificación 3
- Pruebas Realizadas 3

Presentación del software **3**

Resultados **3**

Conclusiones **4**

Introducción

Para está presente investigación para el curso de Aseguramiento de Calidad, en el Instituto Tecnológico de Costa Rica, Sede de Alajuela se propuso que cada estudiante expusiera una herramienta de pruebas unitarias, o bien analizadores de código, para que de esta manera se pueda buscar mejorar la calidad de los proyectos de software y adicionalmente buscar un ambiente de desarrollo más controlado en el que se asegure tanto al programador, como al usuario final un producto de alta calidad en cuantos se refiere a software, para que de esta manera al tener presentes las mismas herramientas pueden ser útiles para el desarrollo de los proyectos a desarrollar por cada grupo de trabajo, con el fin de facilitar el proceso de chequeo del software, aportar el éxito del un proyecto y que este mismo pueda ser controlado con éxito en cuanto a control de calidad y detección de errores o deficiencias, como es el caso de los analizadores de código.

Descripción de la herramienta

PMD (Programming Mistake Detector) es un analizador de código fuente estático de código abierto que informa sobre los problemas encontrados dentro del código fuente de una aplicación.1 PMD incluye conjuntos de reglas incorporados por defecto y admite la capacidad de escribir reglas personalizadas. No informa errores de compilación, ya que solamente puede procesar archivos fuente bien formados. Los problemas informados por PMD son malos hábitos de programación, y secciones de código fuente ineficiente, que pueden reducir el rendimiento y la capacidad de mantenimiento del programa si se acumulan.2 Es capaz de analizar archivos escritos en los lenguajes Java, JavaScript, Apex y Visualforce, PLSQL, Apache Velocity, XML y XSL.

Características

PMD puede detectar fallas o posibles fallas en el código fuente, tales como:

- Errores posibles : Bloques vacíos de sentencias try/catch/finally/switch
- Código muerto: Variables locales, parámetros y métodos privados no usados.
- Declaraciones if/while vacías.
- Expresiones demasiado complicadas: Declaraciones if innecesarias, bucles for que podrían ser bucles while.
- Código subóptimo: Uso de String / StringBuffer inútil.
- Clases con medidas de complejidad ciclomática alta.
- Código duplicado: El código copiado/pegado puede significar errores copiados/pegados y reduce la capacidad de mantenimiento.

Requerimientos

Los requerimientos de PMD se encuentran los siguientes:

1. Tener una computadora con una instalación de Linux o Windows

2. Tener instalado previamente Eclipse, el IDLE respectivo donde se basaron las pruebas de la herramienta. Para que esta misma instalación de la herramienta sea de una manera más fácil para su utilización.

Ventajas

1. Fácil instalación e implementación
2. No requiere una gran curva de aprendizaje
3. Análisis profundo sobre el código analizado
4. Verificar el correcto funcionamiento del programa
5. Mejorar las prácticas como programador
6. Dar visibilidad de vulnerabilidad en el código
7. Se integra con muchos IDE y herramientas de compilación
8. Sólida red de documentación y soporte

Desventajas

1. Existen una limitada cantidad de lenguajes que pueden hacer uso de PMD
2. Da la visibilidad del error, pero no da sugerencias demasiado específicas

Otros detalles a resaltar para información del grupo

En cuanto respecta a PMD él tiene un detector de copiar/pegar (CPD) es un complemento de PMD que utiliza el algoritmo de búsqueda de cadenas de Rabin, Karp para encontrar código duplicado. A diferencia de PMD, CPD funciona con una gama más amplia de lenguajes, incluidos Java, JavaServer Pages (JSP), C, C ++, Fortran, PHP y C #.

Por otro lado, PMD se publica bajo licencia BSDish, mientras que partes de él están bajo Apache License 2.0 y LGPL.

Metodología

Justificación

Con lo que respecta mi decisión como individuo para la selección de la herramienta PMD, fue que esta misma herramienta no es un software de pruebas o similares, sino es un analizador de código siendo algo diferente de lo que se ha mostrado en el curso, ya que es importante poder recalcar las ventajas y debilidades que pueda tener un código, aunque este sea funcional.

Pruebas Realizadas

Con lo que se refiere a las pruebas realizadas se hicieron sobre un programa que se encarga de determinar puntos en un plano cartesiano y que este mismo se puedan unir esos puntos para que posteriormente se puedan calcular la distancia euclidiana entre cada punto del punto actual al punto (0,0). El pow sirve para elevar el número a una potencia para poder sacar el teorema de pitágoras. A continuación se

adjuntan las clases que se usaron para la realización de la prueba de la herramienta:

Clase MyPoint:

```java
public class MyPoint {
    // (5 puntos) Cree la clase MyPoint que modela un punto en 2D e implementa la siguiente
    // funcionalidad, defina los atributos apropiados:
    private int x;
    private int y;

    // (2 puntos) Constructores:
    //  •MyPoint: define un punto en la posición fija (0,0).
    //  •MyPoint (x: int,y: int): define un punto en la posición x,y
    public MyPoint (){this.x = 0;this.y = 0;}

    public MyPoint (int x, int y){this.x = x;this.y = y;}

    // • Gets y sets definidos como de costumbre
    public int getX() {return x;}

    public void setX(int x) {this.x = x;}

    public int getY() {return y;}

    public void setY(int y) {this.y = y;}

    // •El método getXY  retorna un array con los valores de x (posición 0) y y(posición 1).
    public int[] getXY(){int[] point = new int[]{this.x, this.y};return point;}

    // •toString: retorna el String "(x,y)" donde x,y representa el valor x,y del punto.
    @Override
    public String toString(){
        return "(" + String.valueOf(x) + "," + String.valueOf(y) + ")";
    }

    // •distance(x: int,y: int): devuelve la distancia del punto actual al punto (x,y)
    public double distance (int x, int y){
        double primer_punto = Math.pow(x-this.x, 2);
        double segundo_punto = Math.pow(y-this.y, 2);

        double distance = Math.sqrt(primer_punto + segundo_punto);
        return distance;
```

```java
    }

    // •distance(another MyPoint): devuelve la distancia del punto actual al punto another.
    public double distance (MyPoint another){

        // El pow sirve para elevar el numero a una potencia para poder sacar el teorema de pitagoras
        double primer_punto = Math.pow(another.getX() - this.x, 2);
        double segundo_punto = Math.pow(another.getY() - this.y, 2);

        // sqrt es para sacar la raiz cuadrada
        double distance = Math.sqrt(primer_punto + segundo_punto);return distance;}

    // •distance(): devuelve la distancia euclideana del punto actual al punto (0,0).
    public double distance(){

        // El pow sirve para elevar el numero a una potencia para poder sacar el teorema de pitagoras
        double primer_punto = Math.pow(this.x,2);
        double segundo_punto = Math.pow(this.y,2);

        // sqrt es para sacar la raiz cuadrada
        double distance = Math.sqrt(primer_punto + segundo_punto);
        return distance;
    }
```

Clase MyTriangle:

```java
public class MyTriangle {
    // 5 puntos) Defina la clase MyTriangle que representa un triángulo en 2D definido por los
    // tres puntos de las esquinas (utilizando la clase MyPoint). Defina los atributos apropiados y
    // la composición de clases
    private MyPoint v1;
    private MyPoint v2;
    private MyPoint v3;

    // (2 puntos) Constructores:
    //  •MyTriangle(x1: int,y1: int, x2: int,y2: int, x3: int,y3: int): define un triángulo, primero
    //  creando a partir de los xi y yi tres puntos para definir los vértices del triángulo.
    //  •MyTriangle (v1: MyPoint, v2: MyPoint, v3: MyPoint): define un triángulo utilizando
    //  los tres puntos vi para definir los vértices de este
    public MyTriangle (int x1, int y1, int x2, int y2, int x3, int y3)
    {this.v1 = new MyPoint(x1,y1);this.v2 = new MyPoint(x2, y2);this.v3 = new MyPoint(x3, y3);}

    public MyTriangle(MyPoint v1, MyPoint v2, MyPoint v3){this.v1 = v1;this.v2 = v2;this.v3 = v3;}

    // •toString(): devuelve un String con la siguiente información
    // "MyTriangle[v1=(x1,y1),v2=(x2,y2),v3=(x3,y3)]"
    @Override
    public String toString(){
        return "MyTriangle[v1=(" +String.valueOf(v1.getX()) + "," +  String.valueOf(v1.getY()) + "),v2=(" + String.valueOf(v2.getX())
    }

    // •getPerimeter(): devuelve el perímetro del triángulo
    public double getPerimeter(){

        // Se definenen cada lado del triangulo
        double lado_1 = v1.distance(v2);
        double lado_2 = v1.distance(v3);
        double lado_3 = v2.distance(v3);

        // Se suman los lados del triangulo
        return lado_1 + lado_2 + lado_3;
    }
```

```java
    // •GetType(): devuelve el tipo de triángulo que corresponde a "equilátero" si todos los
    // lados del triángulo son iguales, "isósceles" si dos de los tres lados son iguales, o
    // "escaleno" si los tres lados son diferentes
    public String getType(){
        double lado_1 = v1.distance(v2);
        double lado_2 = v1.distance(v3);
        double lado_3 = v2.distance(v3);

        if(lado_1 == lado_2 && lado_2 == lado_3){return "El triangulo es equilatero";}
        else if(lado_1 == lado_2 || lado_1 == lado_3 || lado_2 == lado_3){return "El triangulo es isoseles";}
        else{return "Es triangulo es escaleno";}
    }
}
```

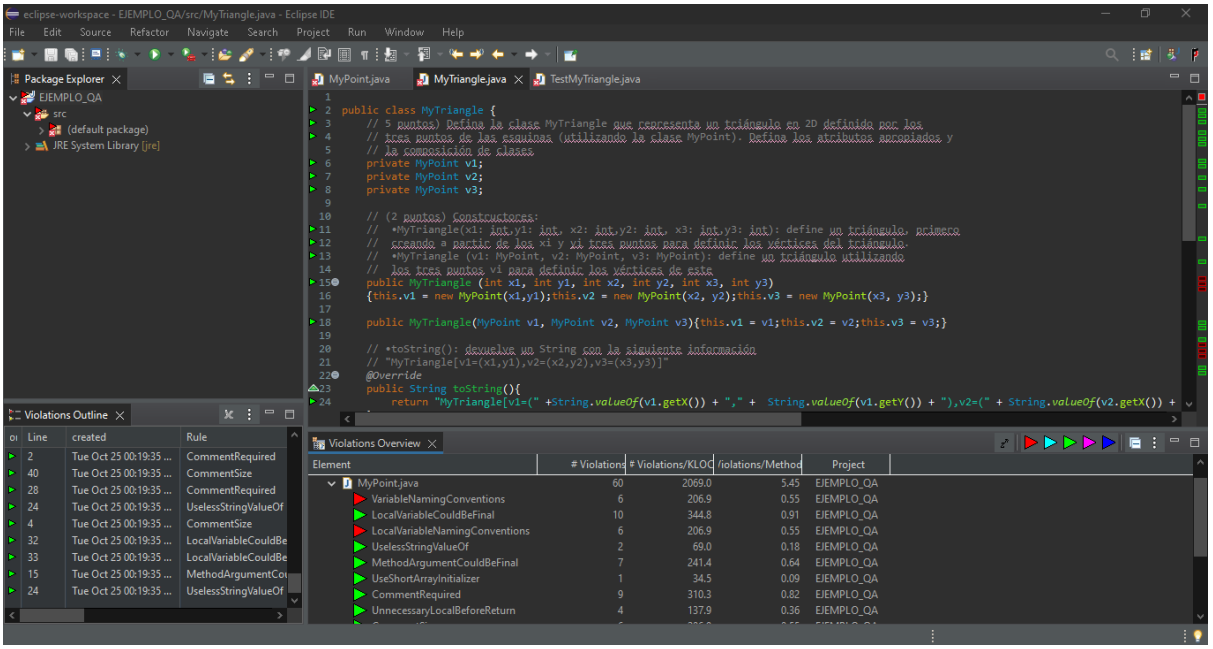
Clase TestMyTriangle:

```java
import java.util.Arrays;

public class TestMyTriangle {
    public static void main(String[] args){

            // •Cree tres puntos
            MyPoint v1 = new MyPoint(5346,53453);
            MyPoint v2 = new MyPoint(1123,7568);
            MyPoint v3 = new MyPoint(12325,542);

            MyTriangle triangulo = new MyTriangle(v1,v2,v3);
            MyTriangle triangulo_1 = new MyTriangle(9,5,1,6,412,20);

            // •Muestre que los métodos de punto funcionan.
            System.out.println(v1.getX());
            System.out.println(v2.getY());

            System.out.println("V2" + v2);
            System.out.println("v2" + v2);

            // •Cree un triangulo a partir de los tres puntos
            System.out.println("V1: " + v1);
            System.out.println("V2: " + v2);
            System.out.println("V3: " + v3);

            System.out.println(v1.distance(14345,14235));
            System.out.println(v2.distance(v3));

            // •Cree un triangulo a partir de valores x,y de tres ubicaciones.
            System.out.println(Arrays.toString(v1.getXY()));
            System.out.println(Arrays.toString(v2.getXY()));
            System.out.println(Arrays.toString(v3.getXY()));

            // •Pruebe con ambos triángulos que los métodos creados funcionan.
            System.out.println(triangulo);
            System.out.println(triangulo_1);
```

```java
            // •Pruebe con ambos triángulos que los métodos creados funcionan.
            System.out.println(triangulo);
            System.out.println(triangulo_1);

            System.out.println(triangulo.getPerimeter());
            System.out.println(triangulo_1.getPerimeter());

            System.out.println(triangulo.getType());
            System.out.println(triangulo_1.getType());

        }
}
```

Presentación del software

Las pruebas realizadas de ejemplo:



Resultados

Los resultados que se tuvieron en las pruebas que se realizaron de la herramienta a través del código expuesto anteriormente se puede notar que la predominancia de buenas prácticas se ve presente sin embargo se pueden ver contemplada una serie de errores no de código, sino de diseño del mismo como tal, siendo un error predominante sobre el manejo de variables que al final se están creando variables en una clase específica la cual esta misma no se está utilizando en la misma tarea.

Conclusiones

En cuanto a la conclusión, el analizador de código PMD fue bastante sustancial para qué desarrollo de las pruebas que se realizaron en el proyecto de prueba, donde esta misma se puede mostrar las deficiencias que pueda tener un código como tal, pudiendo dar descripciones y comentarios de lo que está mal y lo que está bien en el código, si bien es cierto que la instalación de la herramienta es sencilla las descripciones pueden llegar a ser de gran utilidad ya que le informa al usuario sobre sus prácticas como programador, para que de esta manera el programador como tal pueda ser retroalimentado para mejorar en su área.