

Calidad, Pruebas y Mantenimiento del Software.

Continuous Assessment Laboratory 2

Ana Cortés Cercadillo¹, Carlos Javier Hellín Asensio², and Daniel Ferreiro Rodríguez³

¹a.cortesc@edu.uah.es, ²carlos.hellin@edu.uah.es, ³daniel.ferreiror@edu.uah.es

22 de Mayo, 2022

Contents

1	Ejercicio 1	2
1.1	Requisitos	5
1.2	Clonar y generar del Jar	6
1.2.1	Ejecución	7
2	Ejercicio 2	7
2.1	Requisitos	8
2.2	Descargar GitHub	8
2.3	Ejecutar SonarQube	9
2.4	Descargar el Scanner	13
2.4.1	Variable de estado PATH	14
2.5	Resultados de la primera ejecución en el Server	16
2.6	Resultado de la segunda ejecución en el Server	16
2.7	Gráfica Issues	17
2.8	Gráfica Coverage	17
2.9	Gráfica Duplications	18
2.10	Más estadísticas...	18
3	Ejercicios opcionales	18
3.1	Refactoring en Visual Studio Code	18
3.1.1	Extract Method	19
3.1.2	Extract Variable	20
3.1.3	Renombrar símbolo	21
3.2	Análisis de Software Reliability Growth Models	21
3.2.1	Jelinski-Moranda	21
3.2.2	Littlewood-Verrall	22
3.2.3	Musa y Okumoto	22
3.2.4	Goel-Okumoto	22
3.2.5	Weibull	23
3.2.6	Log-Logistic	23

1 Ejercicio 1

El proyecto *CajaRegistradora* empleado en este ejercicio, se crea a partir de la interfaz *ICajaRegistradora* presentada a continuación:

```
1 public interface ICajaRegistradora
2 {
3     /**
4      * @return Nombre de la caja registradora
5      */
6     public String getNombre();
7
8     /**
9      * @return Numero de tipos de moneda que hay en la caja.
10    */
11    public int getNumTipoMonedas();
12
13    /**
14     * @param tipo
15     * @return Devuelve el numero de unidades del tipo de
16     *         moneda pasado como parametro.
17     * -1 en el caso en el que el tipo de moneda no sea
18     *       valido.
19     */
20    public int getUnidadesTipoMoneda(double tipo);
21
22    /**
23     * Ingresa el numero de monedas indicado y del tipo
24     * indicado.
25     * @param tipo
26     * @param unidades
27     * @return true si los parametros son validos, false si
28     *         no
29     */
30    public boolean meterMonedas(double tipo, int unidades);
31
32    /**
33     * Valida un tipo de moneda
34     * @param tipo
35     * @return true si es valida, false si no
36     */
37    public boolean monedaValida(double tipo);
38
39    /**
40     * Extrae de la caja el numero de unidades del tipo de
41     * moneda especificado
42     * @param tipo
43     * @param unidades
44     * @return true si ha sido posible, false en caso
45     *         contrario.
46     */
47    public boolean sacarMonedas(float tipo, int unidades);
48
49    /**
50     * Vacía la caja registradora.
51     */
52 }
```

```

44      */
45      public void vaciarCajaRegistradora();
46
47      /**
48       * @return Devuelve un array con los tipos de monedas que
         tiene.
49      */
50      public double[] getTiposDeMonedas();
51
52      /**
53       * @return Devuelve el saldo actual de la caja
54      */
55      public double getSaldo();
56  }

```

Listing 1: Código

El funcionamiento de la caja registradora se puede comprobar con el siguiente menú:

1. Ingresar monedas.(Pedirá tipo y unidades)
2. Listar contenido.(Por cada tipo de moneda, unidades.)
3. Mostrar saldo.
4. Extraer monedas.(Pedirá tipo y número)
5. Salir.

CK (<https://github.com/mauricioaniche/ck>) calcula métricas de código a nivel de clase y método en proyectos Java por medio de análisis estático (código sin compilar). Este repositorio presenta un gran conjunto de métricas:

- **CBO** (*acoplamiento entre objetos*): cuenta el número de dependencias que tiene una clase. Las herramientas verifican cualquier tipo utilizado en toda la clase e ignora las dependencias del propio Java.
- **CBO Modified** (*acoplamiento entre objetos*): cuenta el número de dependencias que tiene una clase. Esta métrica considera una dependencia de una clase como las referencias que el tipo hace a otros y las referencias que recibe de otros tipos.
- **FAN-IN**: cuenta el número de clases que hacen referencia a una clase en particular.
- **FAN-OUT**: cuenta el número de otras clases referenciadas por una clase en particular.
- **DIT** (*Depth Inheritance Tree*): Cuenta el número de "padres" que tiene una clase.
- **NOC** (*Número de Hijos*): Cuenta el número de subclases inmediatas que una clase particular tiene.
- **Número de campos**: cuenta el número de campos (campos estáticos, públicos, privados, protegidos, predeterminados, finales y sincronizados).

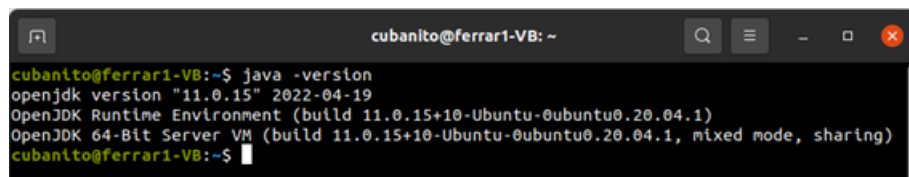
- **Número de métodos:** cuenta el número de métodos (métodos estáticos, públicos, abstractos, privados, protegidos, constructores, predeterminados, finales y sincronizados).
- **Número de métodos visibles:** cuenta el número de métodos visibles (métodos que no son privados).
- **NOSI** (*Número de invocaciones estáticas*): cuenta el número de invocaciones a los métodos estáticos.
- **RFC** (*Respuesta para una clase*): cuenta el número de métodos únicos invocaciones en una clase.
- **WMC** (*Weight Method Class*) o *complejidad de McCabe*: cuenta el número de instrucciones de bifurcación en una clase.
- **LOC** (*Líneas de código*): Cuenta las líneas de código, ignorando líneas vacías y con comentarios.
- **LCOM** (*falta de cohesión de métodos*): calcula la métrica LCOM (no es fiable). LCOM-HS puede ser mejor.
- **LCOM*** (*falta de cohesión de métodos*): esta métrica es una versión modificada de la versión actual de LCOM implementado en CK Tool. LCOM* es una métrica normalizada que calcula la falta de cohesión de clase dentro de un rango de 0 a 1. Entonces, cuanto más cerca de 1 sea el valor de LCOM* en una clase, menor será la cohesión grado de esta clase respectiva y viceversa.
- **TCC** (*Tight Class Cohesion*): mide la cohesión de una clase con un rango de valores de 0 a 1.
- **LCC** (*Loose Class Cohesion*): similar a TCC pero además incluye el número de conexiones indirectas entre clases visibles para el cálculo de la cohesión.
- **Cantidad de returns**
- **Cantidad de bucles:** el número de bucles (es decir, for, while, do while, enhanced for).
- **Cantidad de comparaciones:** el número de comparaciones (es decir, == y !=).
- **Cantidad de try/catches**
- **Cantidad de expresiones entre paréntesis**
- **Cadena:** el número de literales de cadena. Las cadenas repetidas cuentan tantas veces como aparecen.
- **Cantidad de números**
- **Cantidad de operaciones matemáticas**
- **Cantidad de Variables**

- **Máximo de bloques anidados**
- **Cantidad de clases anónimas, clases internas y expresiones lambda**
- **Número de palabras únicas:** El algoritmo básicamente cuenta el número de palabras en un método/clase, después de eliminar las palabras clave de Java. Los nombres se dividen según mayúsculas y minúsculas y subrayado.
- **Número de declaraciones de registro:** número de declaraciones de registro en el código fuente.
- **Tiene Javadoc:** valor booleano que indica si un método tiene javadoc.
- **Modificadores:** modificadores públicos/abstractos/privados/protegidos/nativos de clases/métodos.
- **Uso de cada variable:** con qué frecuencia se usó cada variable dentro de cada método.
- **Uso de cada campo:** con qué frecuencia se usó cada campo local dentro de cada método.
- **Invocaciones de métodos:** todos los métodos invocados directamente, las variaciones son invocaciones locales e invocaciones locales indirectas.

Nota: CK separa las clases comunes de las internas y anónimas. LOC es la única métrica que no está completamente aislada de las demás porque si A tiene una declaración de una clase interna B, entonces $LOC(A) = LOC(\text{clase A}) + LOC(\text{clase interna B})$.

1.1 Requisitos

Tener instalado al menos Java 8 para poder compilar y ejecutar esta herramienta. La versión del **jdk** que se utilizaba es la **11.0.15** (Figura 1).



```
cubantito@ferrari-VB: ~
cubantito@ferrari-VB:~$ java -version
openjdk version "11.0.15" 2022-04-19
OpenJDK Runtime Environment (build 11.0.15+10-Ubuntu-0ubuntu0.20.04.1)
OpenJDK 64-Bit Server VM (build 11.0.15+10-Ubuntu-0ubuntu0.20.04.1, mixed mode, sharing)
cubantito@ferrari-VB:~$
```

Figura 1: Comprobar versión de Java.

Maven solicita la versión de Java (jdk 11) que almacena la variable de entorno `$JAVA_HOME` en Ubuntu. Los siguientes pasos, son una de las tantas formas de hacer persistente una variable de entorno en Linux (Figura 2):

`$ echo $JAVA_HOME` muestra el contenido de la variable y se aprecia que está vacío.

`$ cp /.bashrc /.bashrc.bak` hace una copia de seguridad del archivo `bashrc` (en caso de que se arruine, se puede recuperar).

`echo` se escribe la ruta en donde se encuentra el jdk 11 en el archivo `.bashrc`.

`$ tail -3 /.bashrc` comprueba que se agregó correctamente la línea en cuestión al final del archivo.

\$ `. .bashrc` hace persistente el contenido del archivo `.hashrc`.
 \$ `echo $JAVA_HOME` ahora la variable contiene la ruta en donde se encuentra el jdk 11.

```
cubanito@ferrari-VB: ~$ echo $JAVA_HOME
cubanito@ferrari-VB:~$ cp ~/.bashrc ~/.bashrc.bak
cubanito@ferrari-VB:~$ echo "export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64" >> ~/.bashrc
cubanito@ferrari-VB:~$ tail -3 ~/.bashrc
fi
export PATH=/home/cubanito/sonar-scanner-cli-4.7.0.2747-linux/sonar-scanner-4.7.0.2747-linux/bin:$PATH
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
cubanito@ferrari-VB:~$ . .bashrc
cubanito@ferrari-VB:~$ echo $JAVA_HOME
/usr/lib/jvm/java-11-openjdk-amd64
cubanito@ferrari-VB:~$
```

Figura 2: Comprobar versión de Java.

1.2 Clonar y generar del Jar

Clonar el repositorio de GitHub. Se va dentro de la carpeta `ck` y se ejecuta compila el proyecto para generar un archivo `.jar` en el ordenador (Figura 3).

```
cubanito@ferrari-VB: ~/ck
cubanito@ferrari-VB:~$ git clone https://github.com/mauricioaniche/ck
Clonando en 'ck'...
remote: Enumerating objects: 5654, done.
remote: Counting objects: 100% (708/708), done.
remote: Compressing objects: 100% (237/237), done.
remote: Total 5654 (delta 241), reused 676 (delta 233), pack-reused 4946
Recibiendo objetos: 100% (5654/5654), 883.38 KiB | 1.48 MiB/s, listo.
Resolviendo deltas: 100% (2245/2245), listo.
cubanito@ferrari-VB:~$ cd ck
cubanito@ferrari-VB:~/ck$ mvn clean compile package
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.google.inject.internal.cglib.core.$ReflectUtils$1 (file:/usr/share/maven/lib/guice.jar) to method java.lang.ClassLoader.defineClass(java.lang.String,byte[],int,int,java.security.ProtectionDomain)
WARNING: Please consider reporting this to the maintainers of com.google.inject.internal.cglib.core.$ReflectUtils$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
[INFO] Scanning for projects...
[INFO]
[INFO] com.github.mauricioaniche:ck
[INFO]
[INFO]
[INFO]
```

Figura 3: Clonar el repositorio y compilar el proyecto.

El archivo jar debería estar en una ruta similar a esta:
`/home/cubanito/.m2/repository/com/github/mauricioaniche/ck/0.7.1-SNAPSHOT/`

```
cubanito@ferrari-VB: ~/m2/repository/com/github/mauricioaniche/ck/0.7.1-SNAPSHOT
cubanito@ferrari-VB:~$ cd /home/cubanito/.m2/repository/com/github/mauricioaniche/ck/0.7.1-SNAPSHOT/
cubanito@ferrari-VB:~/m2/repository/com/github/mauricioaniche/ck/0.7.1-SNAPSHOT$ ls
ck-0.7.1-SNAPSHOT.jar          ck-0.7.1-SNAPSHOT.pom          _remote.repositories
ck-0.7.1-SNAPSHOT-jar-with-dependencies.jar  ck-0.7.1-SNAPSHOT-sources.jar
ck-0.7.1-SNAPSHOT-javadoc.jar  maven-metadata-local.xml
cubanito@ferrari-VB:~/m2/repository/com/github/mauricioaniche/ck/0.7.1-SNAPSHOT$
```

Figura 4: Ubicación del archivo jar.

1.2.1 Ejecución

El siguiente comando es super importante y paso a paso se explicarán sus componentes:

```
java -jar ck-x.x.x-SNAPSHOT-jar-with-dependencies.jar  
<project dir> <use jars:true/false> <max files per partition,  
0=automatic selection> <variables and fields metrics?  
True/False> <output dir> [ignored directories...]
```

- *Project dir* se refiere al directorio donde CK puede encontrar recursivamente todo el código fuente (.java) para analizarlo.
- *use jars* busca cualquier archivo .jar en el directorio para resolver mejor los tipos.
- *Max files per partition* le dice a JDT el tamaño del lote a procesar (comienza en 0, pero si ocurren problemas de memoria se debe afinar el valor).
- *Variables and field metrics* indica a CK si también desea métricas a nivel de variable y de campo.
- *output dir* se refiere al directorio donde CK exportará los archivos .csv con las métricas del proyecto analizado.

En el directorio de trabajo, se ejecuta el siguiente comando (Figura 5):

```
java -jar /home/cubanito/.m2/repository/com/github/  
mauricioaniche/ck/0.7.1-SNAPSHOT/ck-0.7.1-SNAPSHOT  
-jar-with-dependencies.jar CajaRegistradora/src false 0 True
```

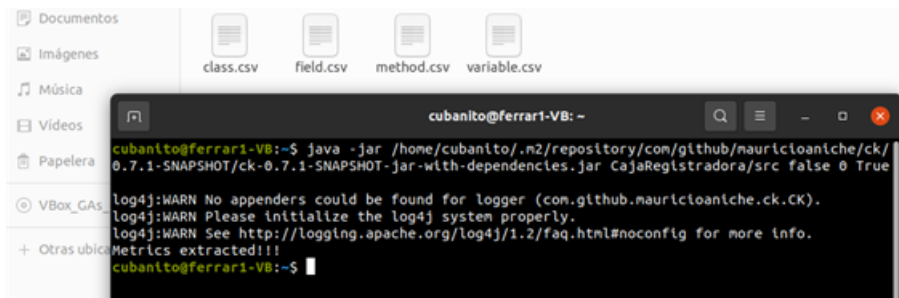


Figura 5: Ejecución del comando.

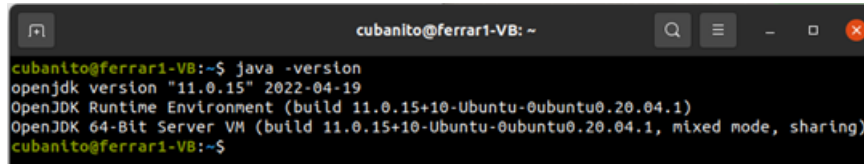
Se genera 4 archivos .csv que se adjuntan en esta práctica con muchas métricas del proyecto *CajaRegistradora*.

2 Ejercicio 2

SonarQube (Sonar) es una plataforma para evaluar código fuente. Software libre que utiliza varias herramientas de análisis estático de código fuente como Checkstyle, PMD o FindBugs para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa.

2.1 Requisitos

Tener instalado la versión del jdk 11 (Figura 6):



```
cubanito@ferrari1-VB: ~  
cubanito@ferrari1-VB:~$ java -version  
openjdk version "11.0.15" 2022-04-19  
OpenJDK Runtime Environment (build 11.0.15+10-Ubuntu-0ubuntu0.20.04.1)  
OpenJDK 64-Bit Server VM (build 11.0.15+10-Ubuntu-0ubuntu0.20.04.1, mixed mode, sharing)  
cubanito@ferrari1-VB:~$
```

Figura 6: Comprobar la versión de Java.

Java 11 es una versión que se puede ejecutar tanto en el Server como el Scanners como se puede ver en la siguiente tabla (Figura 7):

Java	Server	Scanners
Oracle JRE	17	17
	11	11
	8	8
OpenJDK	17	17
	11	11
	8	8

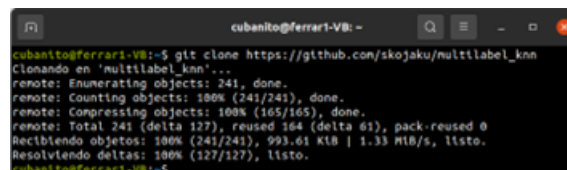
Figura 7: Tabla de versiones.

2.2 Descargar GitHub

`multilabel_knn` es una herramienta escrita en Python para clasificar etiquetas múltiples basadas en los algoritmos del K-vecinos. Se implementan los siguientes algoritmos:

- clasificador k-vecino
- clasificador vecino más cercano multilabel k (recomendado para una pequeña cantidad de etiquetas)
- Clasificador binomial multietiqueta k-vecino más cercano (recomendado para grandes conjuntos de datos con muchas etiquetas)
- Clasificador vecino de gráfico multietiqueta binomial

Clonar el proyecto de GitHub (https://github.com/skojaku/multilabel_knn) (Figura 8).



```
cubanito@ferrari1-VB: ~  
cubanito@ferrari1-VB:~$ git clone https://github.com/skojaku/multilabel_knn  
Clonando en 'multilabel_knn'...  
remote: Enumerating objects: 241, done.  
remote: Counting objects: 100% (241/241), done.  
remote: Compressing objects: 100% (165/165), done.  
remote: Total 241 (delta 127), reused 164 (delta 61), pack-reused 0  
Recibiendo objetos: 100% (241/241), 993.61 KiB | 1.33 MiB/s, listo.  
Resolviendo deltas: 100% (127/127), listo.  
cubanito@ferrari1-VB:~$
```

Figura 8: Clonar repositorio.

Descargar el sonar de la página oficial (<https://www.sonarqube.org/downloads/>) (Figura 9).

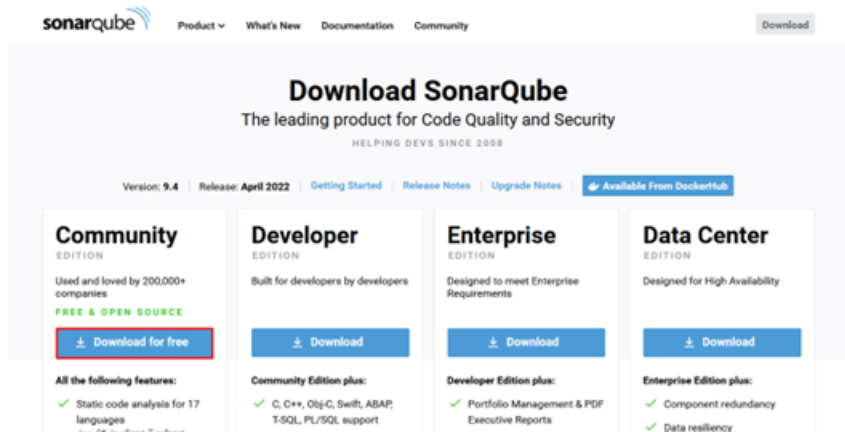


Figura 9: Descargar SonarQube.

2.3 Ejecutar SonarQube

Descomprimir el archivo `$ unzip Descargas/sonarqube-9.4.0.54424.zip`.

En la ruta `./sonarqube-9.4.0.54424/bin/linux-x86-64` se puede encontrar el `sonar.sh` para ejecutar el SonarQube (Figura 10).

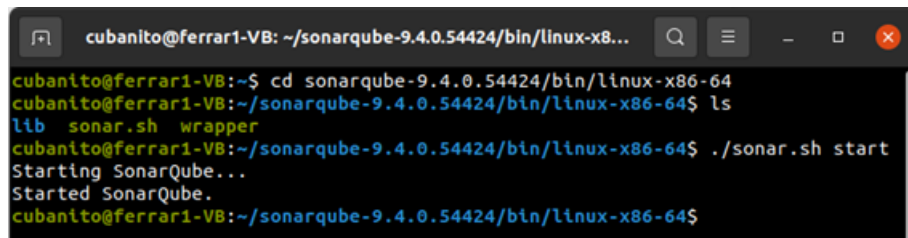


Figura 10: Ejecutar SonarQube.

Abrir el navegador, ir a la página `localhost:9000`. Iniciar sesión con las credenciales `admin` como usuario y contraseña (Figura 11).

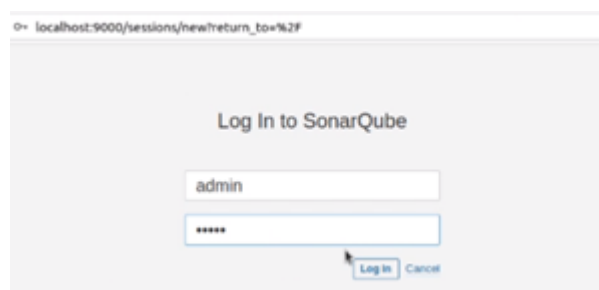


Figura 11: Iniciar sesión.

Luego de loguearse, le pide introducir una nueva contraseña para cambiarla por la contraseña por defecto (Figura 12).

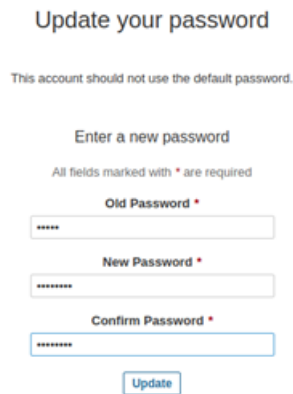


Figura 12: Cambiar contraseña.

Clic en *Add a project* (Figura 13).

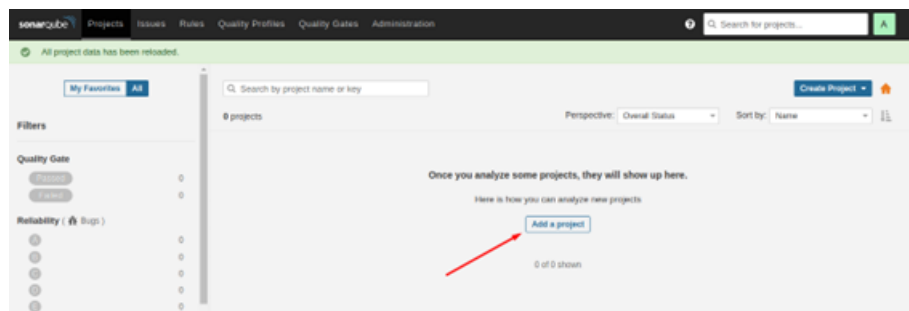


Figura 13: Botón *Add a project*.

Elegir una versión de como se quiere crear el proyecto. En este caso se empleó Manually (manual) (Figura 14).

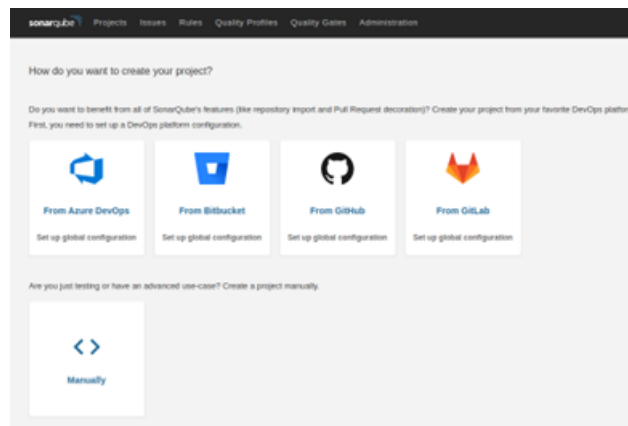


Figura 14: Botón Manually.

Dar nombre al proyecto junto con una clave. Clic en *Set Up* (Figura 15).

The screenshot shows the 'Create a project' form in SonarQube. At the top, a green message bar says 'All project data has been reloaded.' The form has a heading 'Create a project' and a note 'All fields marked with * are required'. There are two input fields: 'Project display name *' and 'Project key *', both containing the text 'multilabel_knn'. Each field has a green checkmark to its right. Below the 'Project key' field, there is a small text block explaining that the project key is a unique identifier and listing allowed characters. At the bottom of the form is a red 'Set Up' button.

Figura 15: Botón *Set Up*.

Se elige la opción *Locally* porque el proyecto está en local (Figura 16).

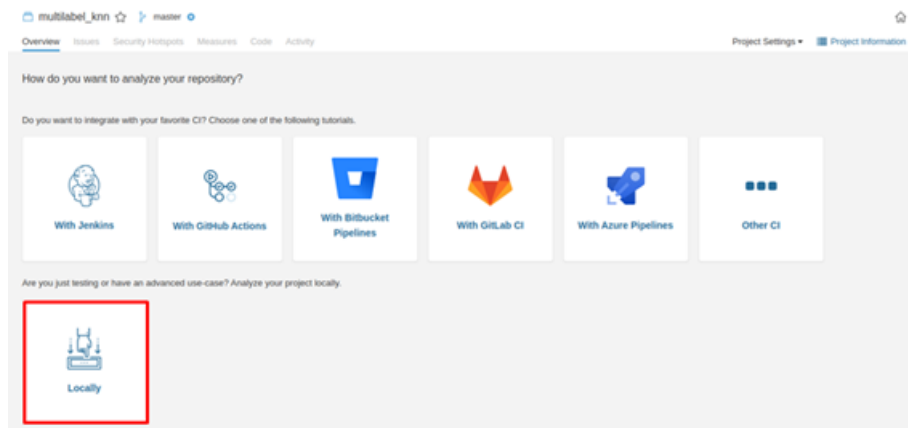


Figura 16: Botón *Locally*.

Se escribe un *token* y clic en *Generate* para crearlo (Figura 17).

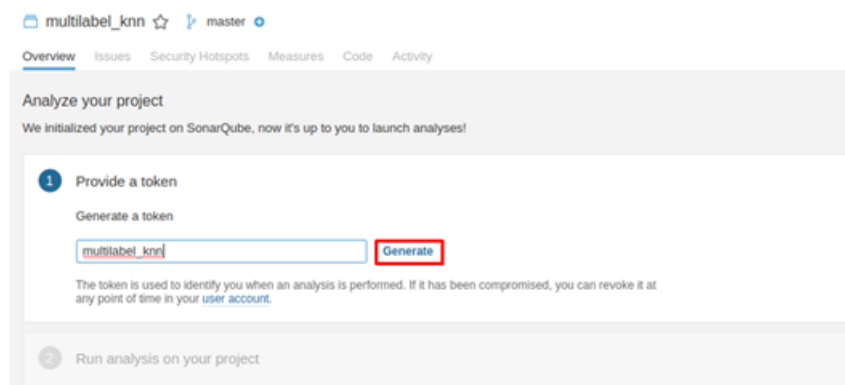


Figura 17: Botón *Generate*.

Clic en *Continue* (Figura 18).

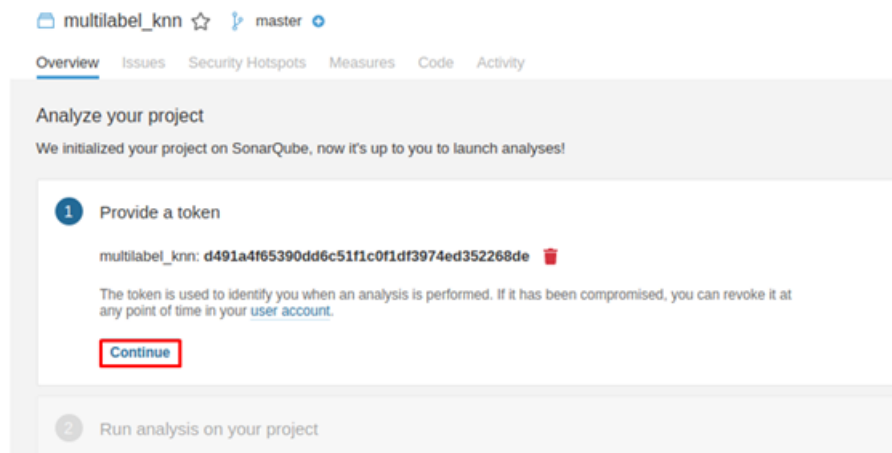


Figura 18: Botón *Continue*.

Elegir la categoría que más se adecua a el proyecto, en este caso es *Other* ya que es un proyecto escrito en Python. Por último, seleccionar el Sistema Operativo (Linux) (Figura 19).

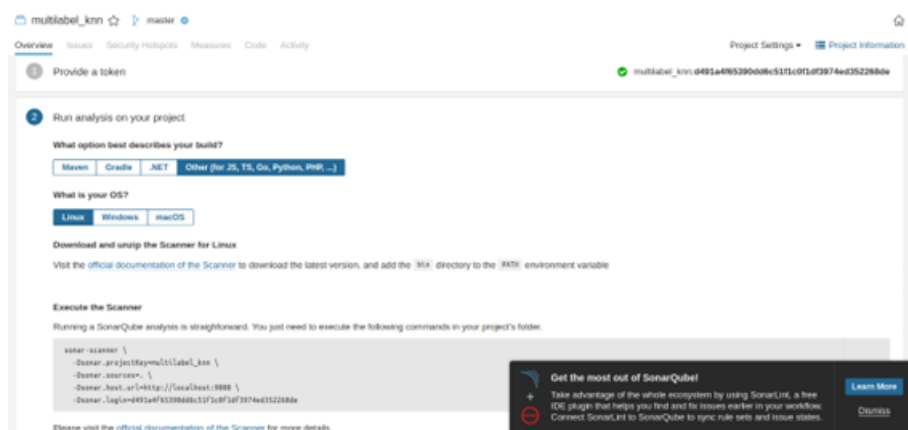


Figura 19: Elegir categoría y Sistema Operativo.

2.4 Descargar el Scanner

Ir a la página que recomienda la aplicación para descargar el SonarScanner (<https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/>) ya que no hay un escáner específico para el sistema de compilación.

Descargar el archivo .zip para su Sistema Operativo (Figura 20).

SonarScanner

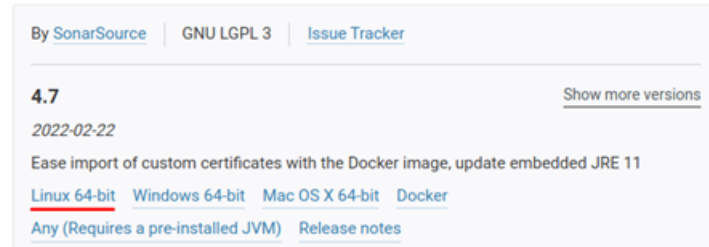


Figura 20: Descargar SonarScanner.

Descomprimir el archivo *SonarScanner* en el directorio de trabajo con el siguiente comando: `$ unzip Descargas/sonar-scanner-cli-4.7.0.2747-linux.zip`

Actualizar la configuración global para que apunte a su servidor SonarQube editando (Figura 21)

`$ directorioInstalacion/conf/sonar-scanner.properties:`

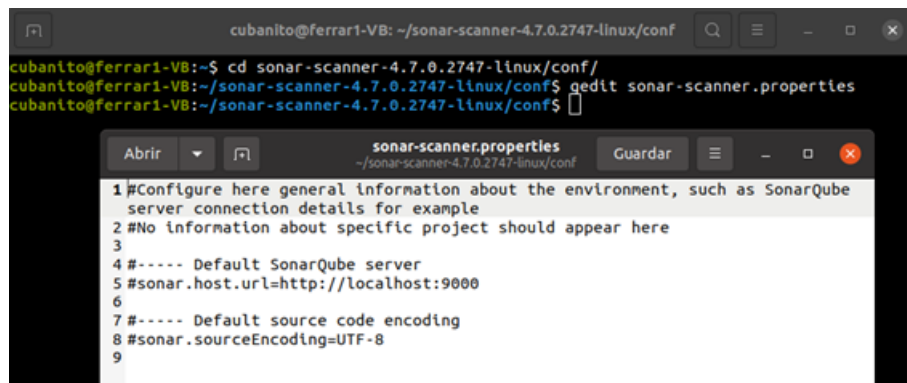


Figura 21: Actualizar la configuración global.

2.4.1 Variable de estado PATH

Añadir el `$ directorioInstalacion/bin` en la variable de estado PATH.

Abrir el archivo `.bashrc` (Figura 22), se escribe en la instrucción `export PATH=/home/cubanito/sonar-scanner-4.7.0.2747-linux/bin:$PATH`

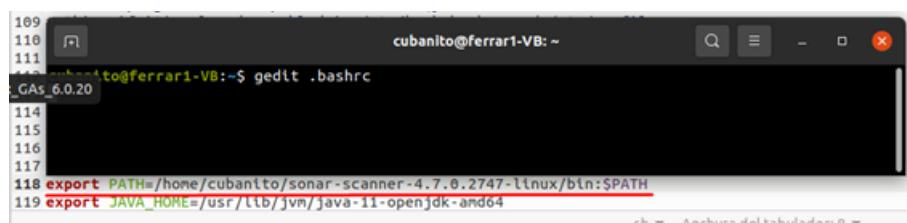



Figura 22: Archivo `.bashrc`.

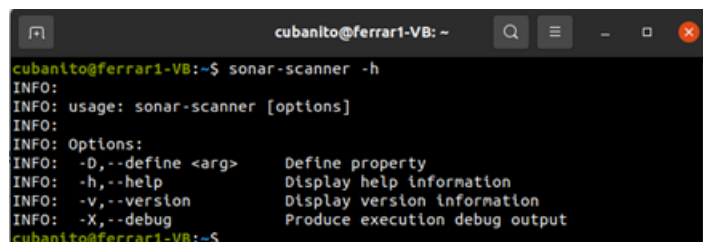
Guardar y cerrar el archivo `.bashrc`. Cierre y vuelva a abrir un terminal o use `$. .bashrc` para guardar el archivo (Figura 23).



```
cubanito@ferrari1-VB: ~  
cubanito@ferrari1-VB:~$ gedit .bashrc  
cubanito@ferrari1-VB:~$ . .bashrc  
cubanito@ferrari1-VB:~$ echo $PATH  
/home/cubanito/sonar-scanner-4.7.0.2747-linux/bin:/home/cubanito/sonar-scanner-cli-4.7.0.2747-linux/sonar-scanner-4.7.0.2747-linux/bin:/home/cubanito/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin  
cubanito@ferrari1-VB:~$
```

Figura 23: Usar `$. .bashrc`.

Comprobar la instalación abriendo un nuevo shell y ejecutando el comando (`sonar-scanner -h`). Debería obtener una salida como esta (Figura 24):



```
cubanito@ferrari1-VB: ~  
cubanito@ferrari1-VB:~$ sonar-scanner -h  
INFO:  
INFO: usage: sonar-scanner [options]  
INFO:  
INFO: Options:  
INFO: -D,--define <arg>      Define property  
INFO: -h,--help              Display help information  
INFO: -v,--version            Display version information  
INFO: -X,--debug              Produce execution debug output  
cubanito@ferrari1-VB:~$
```

Figura 24: Comprobar el comando.

Por último, antes de ir al navegador, se debe mover al interior de la carpeta del proyecto `multilabel_knn` y ejecutar en la consola las siguientes instrucciones (Figura 25):



```
sonar-scanner \  
-Dsonar.projectKey=multilabel_knn \  
-Dsonar.sources=. \  
-Dsonar.host.url=http://localhost:9000 \  
-Dsonar.login=d491a4f65390dd6c51fc0f1df3974ed352268de
```

Figura 25: Instrucciones.

2.5 Resultados de la primera ejecución en el Server

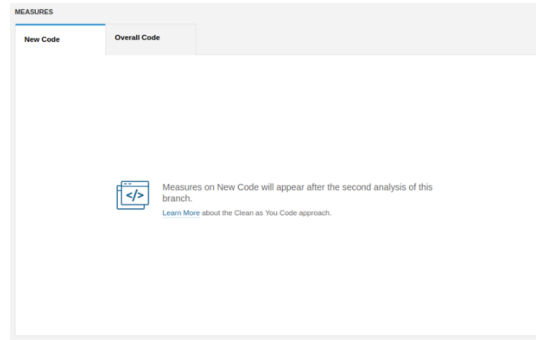


Figura 26

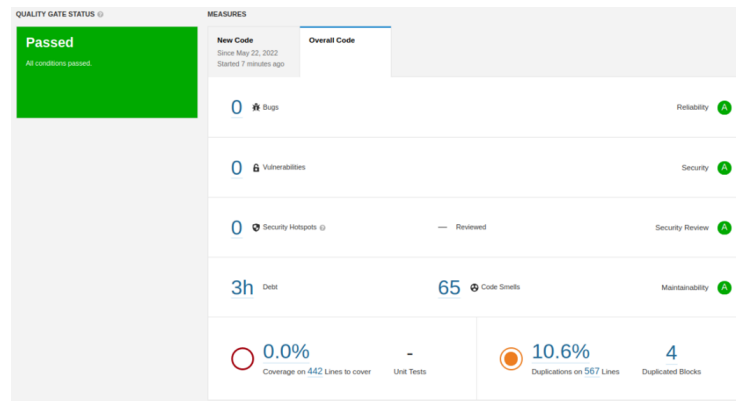


Figura 27

2.6 Resultado de la segunda ejecución en el Server

Se detecta si hay cambios con respecto a la última ejecución, es por eso por lo que los valores son ceros.

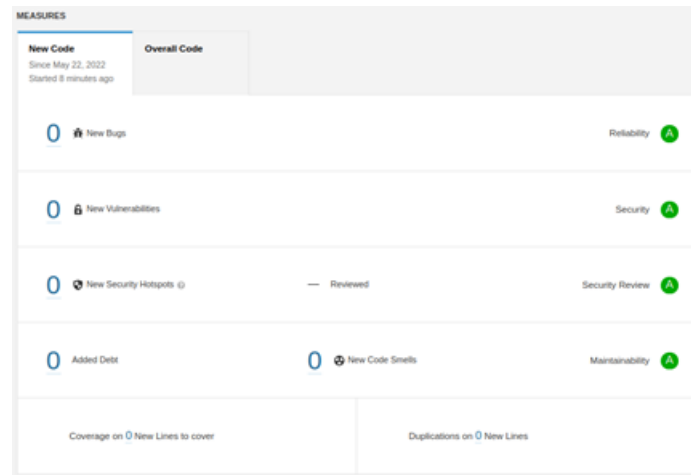


Figura 28

2.7 Gráfica Issues

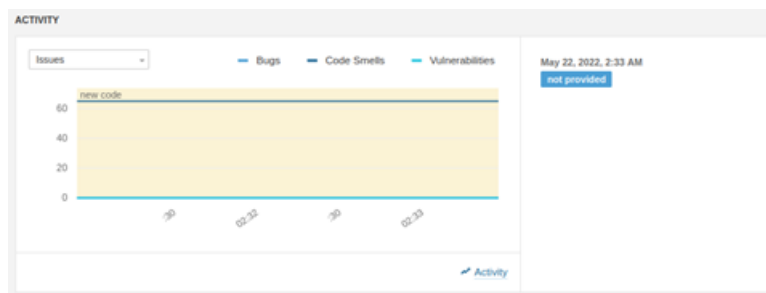


Figura 29

2.8 Gráfica Coverage

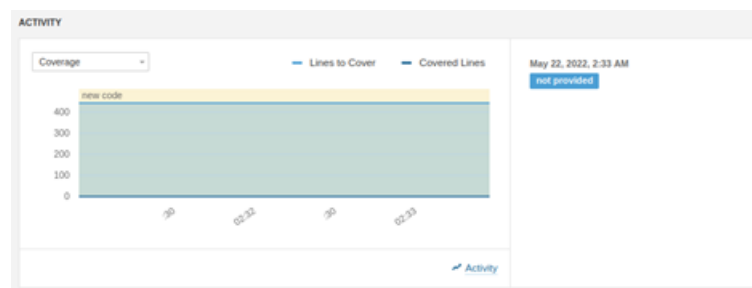


Figura 30

2.9 Gráfica Duplications

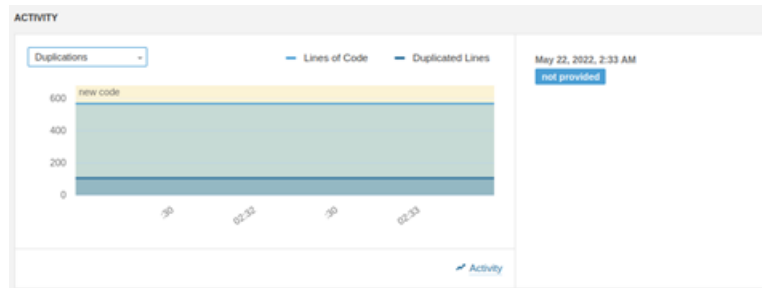


Figura 31

2.10 Más estadísticas...

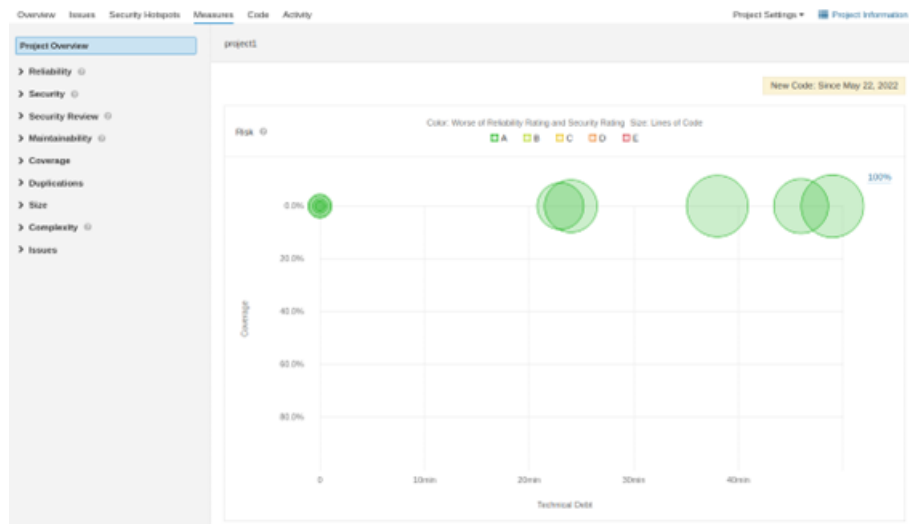


Figura 32

3 Ejercicios opcionales

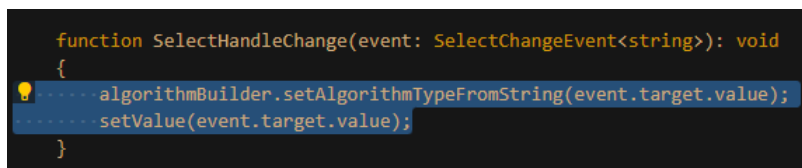
3.1 Refactoring en Visual Studio Code

Se ha elegido como herramienta Visual Studio Code [3] para conocer las operaciones de refactorización que dispone, ya que es un editor muy usado y aporta interés. La refactorización en Visual Studio Code es proporcionando a través de un servicio del lenguaje de programación que se esté usando y, por ejemplo, TypeScript y JavaScript ya tienen soporte incorporado gracias al servicio de lenguaje de TypeScript. En el caso de otros lenguajes de programación, se proporciona este soporte de refactorización a través de las extensiones [4] de Visual Studio Code. Con esto se consigue que, al final, se ofrezca la misma interfaz de

usuario y comandos para la refactorización independientemente del lenguaje de programación.

3.1.1 Extract Method

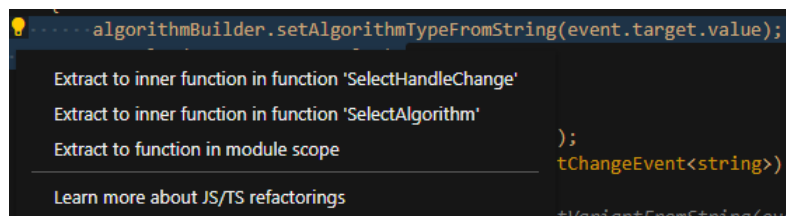
Una refactorización típica es la de evitar la duplicación de código, para ello Visual Studio Code proporciona el Extract Method, en la que se selecciona una zona del código fuente que se quiera reutilizar en otro sitio, y así extraerlo para que esté en su propio método o función.



```
function SelectHandleChange(event: SelectChangeEvent<string>): void
{
  .....algorithmBuilder.setAlgorithmTypeFromString(event.target.value);
  .....setValue(event.target.value);
}
```

Figura 33: Seleccionando una zona del código fuente.

Una vez seleccionado el código a extraer, se puede hacer clic a la bombilla o pulsar `Ctrl + .` para obtener las opciones de refactorización disponibles.

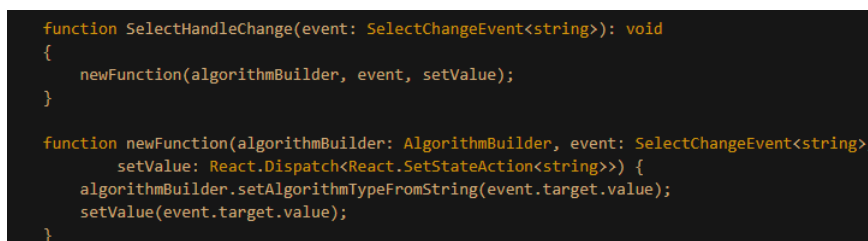


```
.....algorithmBuilder.setAlgorithmTypeFromString(event.target.value);
```

- Extract to inner function in function 'SelectHandleChange'
- Extract to inner function in function 'SelectAlgorithm'
- Extract to function in module scope
- Learn more about JS/TS refactorings

Figura 34: Opciones disponibles de refactorización para este caso.

Como se puede ver en la Figura 35, al elegir la opción "Extract to function in module scope" se ha extraído el fragmento seleccionado a una nueva función en el ámbito del módulo.



```
function SelectHandleChange(event: SelectChangeEvent<string>): void
{
  newFunction(algorithmBuilder, event, setValue);
}

function newFunction(algorithmBuilder: AlgorithmBuilder, event: SelectChangeEvent<string>,
  setValue: React.Dispatch<React.SetStateAction<string>>) {
  algorithmBuilder.setAlgorithmTypeFromString(event.target.value);
  setValue(event.target.value);
}
```

Figura 35: Código extraído en una nueva función.

Por supuesto, existen otras opciones que extraen el código en varios ámbitos diferentes. También, por ejemplo, siguiendo con la Figura 36 se puede extraer código a un método cuando estamos en una clase.

Y como resultado, se tiene un nuevo método privado con el código seleccionado y siendo llamado por el otro método como se puede ver en la Figura 37.

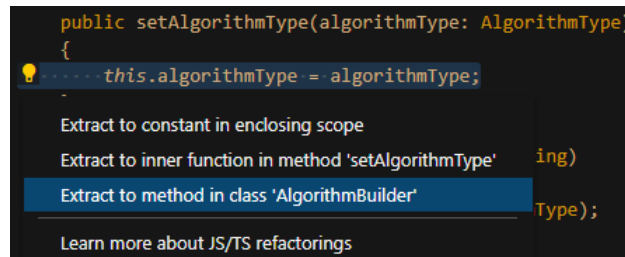


Figura 36: Opciones disponibles al extraer código en una clase.

```
public setAlgorithmType(algorithmType: AlgorithmType)
{
    this.newMethod(algorithmType);
}

private newMethod(algorithmType: AlgorithmType) {
    this.algorithmType = algorithmType;
}
```

Figura 37: Se crea un nuevo método privado.

3.1.2 Extract Variable

Otra de las opciones que proporciona la refactorización en Visual Studio Code, es la posibilidad de crear una nueva variable local para la expresión que ha sido previamente seleccionada, como se puede ver en la Figura 38.

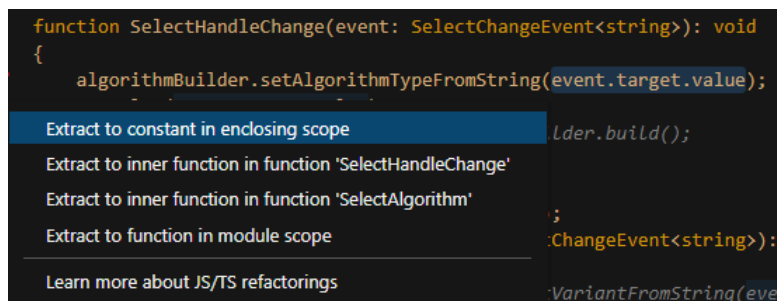


Figura 38: Se selecciona una expresión y se tiene la opción de extraerla como variable local.

Una vez creada, en este caso al tratarse de TypeScript será const y se reemplaza automáticamente por la expresión anterior.

```
function SelectHandleChange(event: SelectChangeEvent<string>): void
{
    const newLocal = event.target.value;
    algorithmBuilder.setAlgorithmTypeFromString(newLocal);
}
```

Figura 39: Resultado de realizar la extracción a variable local

3.1.3 Renombrar símbolo

Por último, se tiene una de las opciones más comunes relacionada con la refactorización del código fuente, que es el renombrado de símbolos. Al seleccionar un símbolo, se pulsa F2, se escribe el nuevo nombre deseado y se pulsa Intro. Todos los usos del símbolo serán renombrados en los distintos ficheros.

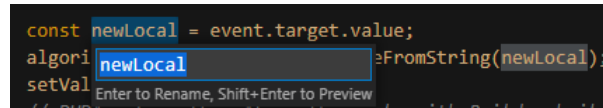


Figura 40: Renombrando uno de los símbolos del código.

3.2 Análisis de Software Reliability Growth Models

Los Software Reliability Growth Models (SRGM) se utilizan para la predicción y estimación de la fiabilidad del software [5]. Estos modelos se basan en una serie de supuestos y estimaciones. Tienen un enfoque estadístico, que necesitan datos de prueba como el tiempo entre fallos o el número de fallos en un intervalo de tiempo [1]. Cada modelo describe cómo observar estos fallos y corregirlos [2].

En esta sección se analiza algunos de estos modelos existentes que intentan modelar el proceso de depuración de una parte del software para poder predecir su fiabilidad futura. Todos ellos intentan modelar unas incertidumbres inherentes en el crecimiento de la fiabilidad del software. Estas incertidumbres pueden describirse como:

- La aleatoriedad de la selección de entradas, es decir, la incertidumbre de cuándo se produce un error.
- La aleatoriedad de la depuración, es decir, la incertidumbre sobre si el error se ha eliminado correctamente, y cuánta diferencia supondrá su eliminación en la fiabilidad global del sistema.

3.2.1 Jelinski-Moranda

El modelo Jelinski-Moranda tiene una complejidad limitada, ya que no tiene en cuenta la aleatoriedad de la incertidumbre de la depuración. Las suposiciones del modelo son las siguientes:

- Los fallos se distribuyen de forma aleatoria.
- Los fallos son independientes y existe una misma probabilidad de causar un fallo durante la prueba.
- El software tiene N fallos iniciales, donde N es desconocido pero fijo.
- Todas las eliminaciones de fallos son perfectas y, además, causan un mismo aumento en el crecimiento de la fiabilidad.
- Los tiempos son variables aleatorias independientes.

[1]

3.2.2 Littlewood-Verrall

El modelo Littlewood-Verrall tiene una mayor complejidad en comparación con el de Jelinski-Moranda. Lo que hace es modelar la aleatoriedad de la incertidumbre de la depuración, introduciendo dos distribuciones aleatorias, una para las incertidumbres de cuándo se produce un error y otra para la incertidumbre sobre si el error se ha eliminado correctamente. Las suposiciones del modelo son las siguientes:

- Los fallos se distribuyen de forma aleatoria.
- No hay un número fijo de fallos
- La eliminación de los fallos es imperfecta.

Este modelo tiene en cuenta la incertidumbre de la depuración y representa tanto el crecimiento como la disminución de la fiabilidad, pero no consigue estimar cuántos son el número de fallos que quedan.

[1]

3.2.3 Musa y Okumoto

El modelo Musa y Okumoto está basado en el proceso de Poisson no homogéneo para datos de tiempo entre fallos. Es un intento de tener en cuenta los fallos pero con diferentes niveles de gravedad. Las suposiciones del modelo son las siguientes:

- El número acumulado de fallos sigue una distribución de Poisson con una media del tipo $m(t)$.
- No hay un número fijo de fallos.
- La detección de fallos se hace de forma independiente.
- El número esperado de fallos es una función logarítmica del tiempo.
- La intensidad de los fallos disminuye exponencialmente con el número esperado de los fallos experimentados

Por norma general, se encuentra que los primeros fallos detectados son los que cuyo arreglo tiene el mayor efecto en cuanto al crecimiento de la fiabilidad, por lo tanto, las correcciones posteriores tienen un efecto menor que las primeras.

[1]

3.2.4 Goel-Okumoto

El modelo Goel-Okumoto se basa en el de Jelinski-Moranda, donde el parámetro N se trata como una variable aleatoria de Poisson con media m . Esto es que describe la tasa de detección de fallos, como un proceso de Poisson no homogéneo, asumiendo que la tasa de peligro es proporcional al número de fallos restantes.

[1]

3.2.5 Weibull

El modelo Weibull es también conocido como el modelo Goel-Okumoto generalizado, ya que en cuenta un aspecto más de la vida real, que lo tradicionalmente hace Goel-Okumoto. En las pruebas ha habido una tendencia a que los proyectos aumenten la intensidad de los fallos desde el principio de la fase de pruebas y, luego más tarde, disminuyen a medida que el equipo de programadores mejora en la búsqueda y corrección de estos errores. Esto es algo que se omitió en el modelo original de Goel-Okumoto, pero que se ha incorporado dentro del modelo de Weibull.

[1]

3.2.6 Log-Logistic

El modelo Log-Logistic abre una posibilidad de que la tasa de ocurrencia de fallos aumente y disminuya a lo largo del tiempo y en el mismo conjunto de datos. Esta característica no es posible en el caso del modelo Weibull. Esto, sin embargo, es debido a la forma del modelo que suele aplicarse mucho mejor para hardware que para software.

[1]

References

- [1] Lokendra K. Sharma, R. K. Saket, and B. B. Sagar. “Software Reliability Growth Models and tools - a review”. In: *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*. 2015, pp. 2057–2061.
- [2] Rashmi Upadhyay and Prashant Johri. “Review on Software Reliability Growth Models and Software Release Planning”. In: *International Journal of Computer Applications* 73 (July 2013), pp. 1–7. DOI: 10.5120/12790-9769.
- [3] *Visual Studio Code*. <https://code.visualstudio.com/> [Último acceso 15/mayo/2022].
- [4] *Visual Studio Code: Extension Marketplace*. <https://code.visualstudio.com/docs/editor/extension-marketplace> [Último acceso 15/mayo/2022].
- [5] Jintao Zeng et al. “A Prototype System of Software Reliability Prediction and Estimation”. In: *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*. 2010, pp. 558–561. DOI: 10.1109/IITSI.2010.90.