Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Trabajo Fin de Grado

Elaboración de ejemplos de aplicación de las técnicas de aseguramiento de calidad del software

Autor: Estefanía Martín Rojas

Tutor/es: Luis Fernández Sanz

2021

UNIVERSIDAD DE ALCALÁ

Escuela Politecnica

GRADO EN INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Trabajo Fin de grado

Elaboración de ejemplos de aplicación de las técnicas de aseguramiento de calidad del software

Autor: Estefanía Martín Rojas

Tutor: Luis Fernández Sanz

TRIBUNAL:

Presidente: …….

Vocal 1º: …….

Vocal 2º: …….

CALIFICACION: ……

FECHA: ……

Agradecimientos

Doy las gracias a mi familia y a mi pareja por haberme apoyado y ayudado desde el primer día de carrera. Gracias a mis compañeros a los que fui conociendo a lo largo de estos años y me han ayudado en infinidad de ocasiones y que a día de hoy siguen haciéndolo.

Agradecer a la Universidad de Alcalá de Henares y a los profesores ya que, sin lo aprendido estos años, no podría involucrarme como lo hago en mi actual empleo y en los que vendrán.

Todos me ayudaron a interesarme por la carrera y a seguir adelante con más ilusión año tras año.

Y, por último, especial gracias a mi tutor Luis Fernández Sanz, que incluso con el periodo caótico de la pandemia ha sido capaz de ayudarme a realizar este trabajo. Le agradezco que me diera la oportunidad de realizar este trabajo y poder profundizar sobre el tema que más me ha interesado en los últimos años de carrera y que tengo intención de seguir explorando: la Calidad del Software.

“Nada con excesos, todo con medida.”

Solón.

Índice

Índice de figuras y tablas

FIGURAS

TABLAS

Resumen

En este proyecto se presentarán estudios, pruebas y usos de seis herramientas gratuitas disponibles en línea (salvo una) y útiles para diferentes etapas y aspectos del ciclo de vida de un proyecto de software. El objetivo es mostrar el funcionamiento y la utilidad de estas herramientas que pueden resultar interesantes para cualquier estudiante o profesional que se encuentre en el ámbito de la ingeniería del software. Se enmarcan en procesos relacionados con la ingeniería inversa en bases de datos, métricas de software y pruebas y estudios del código. Se mostrarán pruebas y manuales de uso de cada una de ellas.

Palabras clave: ingeniería del software, ingeniería inversa, pruebas unitarias, SonarQube, herramientas online.

This project will present studies, tests, and uses of six free tools available online (except one) which are useful for different stages and aspects of the life cycle of a software project. The goal is to show the operation and usefulness of these tools that may be of interest to any student or professional in the field of software engineering. They are framed in processes related to reverse engineering in databases, software metrics and tests and studies of code. Tests and user manuals for each of them will be shown.

Keywords: software engineering, reverse engineering, unit testing, SonarQube, online tools

---

Capítulo 1. Memoria

1.1 Presentación del tema

La Ingeniería del Software es una disciplina que permite mejorar un producto de software gracias a las medidas que nos ofrece para controlar la calidad y los resultados de las actividades de desarrollo. En ocasiones, estas medidas pueden ser costosas de calcular si se trata de una amplia cantidad de código y esto puede, incluso, llevar a errores. Lo óptimo es poder utilizar herramientas que nos permitan visualizar medidas que encontremos oportunas sin perder tanto tiempo en calcularlas nosotros mismos.

Sin embargo, no todas las herramientas resultan fáciles de instalar o interpretar, en especial para estudiantes nuevos en asignaturas de Calidad del Software. Para este tipo de estudiantes, se fundamental encontrar herramientas simples que ayuden a entender conceptos básicos como complejidad ciclomática o ingeniería inversa, así como el uso de scripts y otras técnicas.

La mayoría de los estudiantes necesitan resultados rápidos y claros sin necesidad de perder demasiado tiempo en ello ni estar atentos a costes de licencias o de instalación. Esto es igualmente aplicable a cualquier usuario que quiera comparar resultados con otras herramientas o que no posea suficientes medios para realizar un proyecto completo de software.

En este trabajo se tratará de demostrar cómo este tipo de herramientas pueden solucionar aspectos como la obtención de resultados claros, concretos y rápidos, funcionamiento e interfaces intuitivas amplia lista de lenguajes de programación soportados, etc. La gran mayoría de herramientas que se presentan se encuentran en línea, ya que resulta muy atractivo poder acceder a ellas en cualquier momento desde cualquier dispositivo. Sin embargo, es necesario recalcar que, a pesar de no existir un coste monetario en ninguna de ellas, habrá otros tipos de costes como limitaciones en el tamaño del código soportado, la longitud de los scripts, limitaciones en el uso o en que los resultados se muestren de manera pública, etc.

1.2 Objetivos

Se plantean seis herramientas online (a excepción de una), que corresponden con diferentes aspectos presentes en los procesos de la ingeniería del software, que son:

- **Ingeniería inversa:** dbDiagram.io y diagrams.net
- **Recuperación de diseño mediante diagrama de flujos a partir de código:** Code2Flow y Code Visual to Flowchart (no online).
- **Medidas de software:** Lizard.
- **Análisis del código:** SonarQube.

A partir de scripts encontrados en webs o realizados por mí, se mostrará en cada una de ellas:

- Instalación y/o integración (si la hubiere).
- Funcionamiento de la interfaz.
- Demostración de los resultados con los ejemplos, en ocasiones en diferentes lenguajes de programación.
- Alcance.
- Problemas encontrados.

- Diferencias encontradas en el caso de analizar más de una herramienta.

En ningún caso:

- Se mostrará el funcionamiento de todas las herramientas que se han encontrado y revisado inicialmente sobre el tema en cuestión: solo se mencionarán al final del capítulo.
- Los scripts y código no serán grandes: se usarán scripts y código cortos que resulten didácticos para mostrar la herramienta y sus resultados.
- No se mostrarán las características que podrían obtenerse con cuentas premium si las hubiere.
- No se utilizarán plugins de las herramientas en IDE como VisualStudio, NetBeans, Eclipse, etc Solo se muestra las opciones de servicio en línea.

1.3 Glosario de términos

1.3.1 Términos específicos

Cobertura: es una medida del grado en que una prueba ejercita algunas funciones o código.

Complejidad ciclomática: Se trata de una medida útil para pruebas de caja blanca e identificar la complejidad que presentan las líneas de código de un programa.

Fiabilidad: probabilidad numérica condicional a un nivel de confianza dado, de que los componentes o sistemas realizarán sus funciones previstas sin fallas o satisfactoriamente y dentro de un período de tiempo, función, período o tiempo de misión especificado, cuando se utilicen de la manera y para el propósito previsto mientras operando bajo la aplicación especificada y los niveles de estrés de operación.

Ingeniería inversa: proceso que se realiza para conseguir información a partir de un producto final, de manera que se pueda conocer cómo está diseñado, de qué está compuesto, cómo funciona y cómo realizar su fabricación.

Mantenibilidad: la probabilidad de que un artículo fallido se restaure a su estado operativo satisfactorio.

Mantenimiento: todas las acciones necesarias para retener un artículo o equipo en, o restaurarlo a, una condición especificada.

Seguridad: número vulnerabilidades encontradas en el código.

1.3.2 Acrónimos

ABAP: Advanced **B**usiness **A**pplication **P**rogramming
API: Application **P**rogramming **I**nterface
COBOL: Common **B**usiness-**O**riented **L**anguage
DNS: Domain **N**ame **S**ystem
IDE: Integrated **D**evelopment **E**nvironment
ISO: International **O**rganization for **S**tandardization
JS: Java**S**cript
LOC: Lines **O**f **C**ode
REST: Representational **S**tate **T**ransfer

SQL: Structured **Q**uery **L**anguage
TS: Type**S**cript
UML: Unified **M**odeling **L**anguage

1.4 Estructura del trabajo

Los capítulos están estructurados de esta manera:

El **capítulo 2** mostrará el uso de dos herramientas con las que se puede realizar **ingeniería inversa en bases de datos**, en concreto, las que los usuarios pueden usar para crear un modelo de datos a partir de código. Estas herramientas son: **dbDiagram.io y diagrams.net**, ambas realizan las mismas tareas y se mostrarán sus diferencias y su funcionalidad.

En el **capítulo 3** se mostrarán dos herramientas **Code2Flow y Code Visual to FlowChart**. Esta última es instalable en el equipo, pero, a pesar de tener una versión reducida gratis, su aportación es suficiente para el objetivo de este capítulo. Aquí se tratará de crear diagramas de flujo a partir de código o pseudocódigo.

El **capítulo 4** tratará sobre mediciones en el software como complejidad ciclomática, Hasltead y enumeración de líneas de código. Este capítulo solo tratará con una herramienta llamada **Lizard** que muestra en pantalla los resultados de estas medidas a partir de un código. A partir de cálculos hechos a mano, se demostrará la justificación de cada medida obtenida por Lizard.

Por último, el **capítulo 5** es el más extenso y trata sobre la herramienta **SonarQube** y su instalación, funcionamiento y uso como aplicación web **SonarCloud**. Para esto, se ha utilizado un proyecto en Java y otro en Python subidos a GitHub. Para la demostración de los resultados sobre el código, también se han utilizado proyectos públicos de otras personas encontrados en esta aplicación.

Capítulo 2. Ingeniería inversa en bases de datos

2.1 Introducción.

En este apartado trataremos con herramientas que ofrezcan la posibilidad de poder crear código SQL a partir de un esquema relacional de una base de datos y viceversa. Cabe destacar que las herramientas exportan el código compatible para diferentes clientes de bases de datos como pueden ser MySQL, PostgreSQL, Oracle, Microsoft SQL, etc. Incluso, en ocasiones, también se podrá exportar en PNG o PDF. Todo se podrá realizar de manera online, sin necesidad de instalar nada. En este documento se probará el ejemplo en los dos primeros clientes mencionados: MySQL y PostgreSQL.

2.2 Ingeniería inversa.

La ingeniería inversa es el proceso que se realiza para conseguir información a partir de un producto final, de manera que se pueda conocer cómo está diseñado, de qué está compuesto, cómo funciona y cómo realizar su fabricación.

La ingeniería del software se desarrolla en etapas de diseño, desarrollo e implementación. El producto final es el resultado de todas estas etapas. Si se desconoce todo el conocimiento sobre el producto, habría que hacer uso de la ingeniería inversa para poder realizar modificaciones o mantenimiento en este. [2]

Las ventajas que otorga la ingeniería inversa en cualquier caso pueden ser [1]:

- No cambia la funcionalidad original del producto final.
- Al conocer mucho sobre el producto, se mejora el mantenimiento y las complejidades que puedan existir se reducen.
- Se generan otros puntos de vista, como, por ejemplo, crear un esquema a partir de código fuente. Esto facilita la creación de nuevas alternativas.
- Recuperar información perdida y actualizarla.
- Detección de efectos laterales, debido a los cambios que se realicen.
- Reutilización, ya que puede darse el caso de poder utilizar métodos o componentes de otros productos existentes y conocidos, de manera que el desarrollo sea más sencillo, aumentando la productividad y reduciendo los costes.

2.2.1 Ingeniería inversa en base de datos.

Permite conocer la representación conceptual del esquema de una base de datos a partir de su codificación. De esta manera, en el caso de pérdida de información, acerca del proyecto, se puede redocumentar, reconstruir e incluso actualizar la información que ya se tenía.

La información que se extrae de aquí está relacionada con la estructura de la base de datos como entidades, relaciones, atributos, claves primarias… etc. [3]

2.3 Ejemplo para implementar.

Para demostrar el funcionamiento de las herramientas, se utilizará un modelo ya creado en PgModeler, una herramienta para crear modelos relacionales creada para PostgreSQL. El modelo fue creado como enunciado para la realización de una práctica en la asignatura de Bases de Datos Avanzadas en la universidad de Alcalá de Henares y consiste en una gestión de grupos musicales (ver Figura 1):

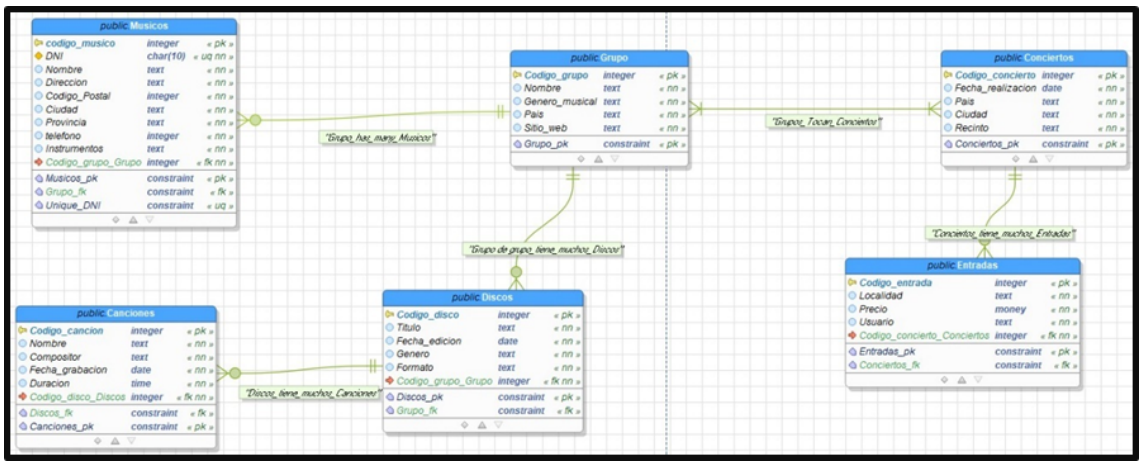


Figura 1. Ejemplo para implementar en pgModeler.

Los **grupos** están formados por **músicos** (entre uno a diez personas), y han creado mínimo un disco. Los **discos** contienen una media de 12 **canciones**. Además, todos los grupos han realizado al menos diez **conciertos** y todos los conciertos deben estar asociados a un grupo musical (hay un millón de músicos y 100.000 conciertos). Las **entradas** se distribuyen de manera aleatoria por todos los conciertos (hay 24 millones de entradas a repartir).

Estos datos son necesarios para las cardinalidades en los modelos que a continuación se crearán.

2.4 dbDiagram.io

Se trata de una herramienta online en inglés que permite crear diagramas de entidad-relación. Esta web permite crear desde cero o a partir de un código en SQL de clientes como MySQL, PostgreSQL, SQL Server o Rails.

Será necesario crear una cuenta en la web a partir de una cuenta en Google o GitHub para poder guardar los proyectos que se vayan creando.

2.4.1 Pantalla principal

La pantalla principal tiene este aspecto:

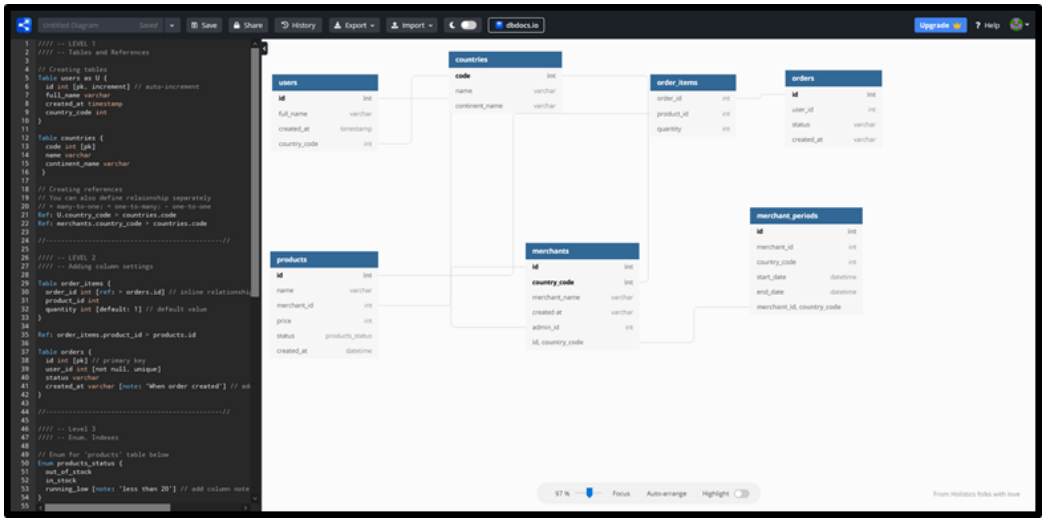


Figura 2. Pantalla principal en dbDiagram.

En la parte izquierda de la pantalla se escribirá la estructura del diagrama, escribiendo las tablas que lo componen y sus atributos y relaciones. Al mismo tiempo, en la parte derecha, se podrá observar en tiempo real cómo se va formando a partir del código de la izquierda, el diagrama de entidad relación.

En la parte superior se podrán distinguir diferentes opciones que ofrece esta herramienta.

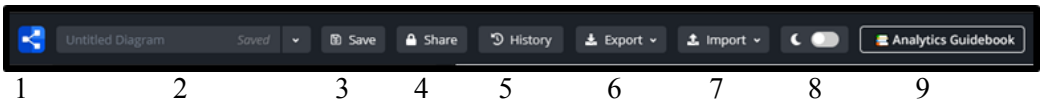


Figura 3. Elementos en la barra superior de la pantalla principal.

1. **Símbolo de la web:** Si se selecciona este botón, volverá a la página principal de la web.
2. **Barra del proyecto:** En este espacio se escribe el nombre del proyecto que se está realizando. Además, en la parte derecha informa si está o no salvado el proyecto. También en el botón más extremo derecho permite crear otro proyecto, abrir uno existente o abrir un ejemplo.
3. **Salvar:** Para guardar el proyecto. Será necesario crear una cuenta en la web.
4. **Compartir:** Permite compartir el proyecto a partir de un enlace con o sin contraseña. Por ejemplo, el proyecto que se creará a continuación será este: https://dbdiagram.io/d/5fdf95a99a6c525a03bbc2cb.
5. **Historial:** Aquí se pueden consultar las versiones que, hasta ahora, se han realizado en el proyecto.
6. **Exportar:** El proyecto se podrá exportar en diferentes formatos:
    a. PDF.
    b. PostgresSQL.
    c. MySQL.
    d. SQL Server.

e. PNG.

7. **Importar:** Si tenemos ya un proyecto creado en otro cliente, se puede importar el proyecto a partir de su SQL. Hay que recordar que solo es necesario el código relacionado con la estructura de las tablas, no es necesario opciones adicionales como el tratamiento de las búsquedas o los índices.
8. **Modo nocturno:** La parte derecha de la pantalla donde se encuentra dibujado el diagrama tendrá un fondo oscuro. Sin embargo, no es posible para cuentas gratis, para poder activarlo habrá que publicar un tweet o apuntarse al plan de pago de la página web.
9. **Otros productos de la web:** Apartado muestra otras herramientas que ofrece la web.

En la parte derecha de la barra superior, se encuentra la documentación oficial de la herramienta seleccionando el botón "Help". El enlace de la documentación es: https://www.dbml.org/docs.

2.4.2 Creación de tablas

A la hora de crear las tablas hay que seguir la siguiente nomenclatura (comillas incluidas):

```
Table "Nombre_tabla" {
      "nombre_atributo" tipo_variable [tipos_de_atributo]
      …
}
```

- **Nombre de la tabla:** Si el nombre presenta más de una palabra, debe estar escrito entre comillas dobles.
- **Atributos** [4]**:** Formados por:
  - El nombre del atributo aparece como "nombre_atributo"
  - El tipo de variable del atributo puede ser cualquiera siempre que sea una sola palabra, por ejemplo, integer, char(x), decimal (1,2) …
  - La configuración del atributo [5] estará rodeada por corchetes. Los valores disponibles son:
    - primery key/pk
    - null/not null
    - unique
    - default: valor [6]. En este caso, si es un valor numérico no es necesario las comillas. Si se trata de una cadena de caracteres, debe estar rodeado por comillas simples. Para valores booleanos, puede tener valor true, false o null. También se permiten expresiones.
    - Increment: incremento automático.
  - Se pueden añadir comentarios en cualquier línea con "Note: "(sin comillas)

2.4.3 Importar a partir de SQL.

En la barra superior en la pantalla principal, seleccionando el botón "import" ofrece diferentes clientes para importar su código SQL. Cuando se ha seleccionado uno de ellos, habrá dos maneras de importar el código, en un espacio para escribir el script o importando el script de manera externa.

En este documento se probará con los clientes PostgreSQL y MySQL el código creado a partir del diagrama que se indica en el apartado 3.3.

La importación de código a partir de PostgreSQL es muy cómoda, no requiere modificar el original a no ser que una línea que contenga la configuración "ALTER TABLE" en una tabla vaya seguida del usuario dueño de esa tabla y no haya ninguna modificación por parte de la configuración. De ser así, solo habría que borrar la opción.

En el caso de MySQL es menos intuitivo ya que no se respeta la nomenclatura de los scripts de este cliente. Es importante saber que en el script hay que remover todas las comillas que puedan encontrarse en el documento, tanto dobles como simples. También retirar todo lo relacionado con los índices que puedan existir en las tablas, quedando de esta manera:

```
CREATE TABLE IF NOT EXISTS mydb.CONCIERTOS (
  codigo_concierto INT NOT NULL,
  fecha_realizacion DATE NULL,
  pais TEXT NULL,
  ciudad TEXT NULL,
  recinto TEXT NULL,
  PRIMARY KEY (codigo_concierto))
ENGINE = InnoDB;
```

Finalmente, una vez se haya subido el nuevo código, la web adaptará el código de manera que se pueda compilar correctamente en su IDE, esto tiene como consecuencia que el script que se haya utilizado cambiará a partir de hora mientras se esté utilizando la herramienta. Si se desea tener el código modificado de nuevo disponible para las herramientas anteriores mencionadas, se deberá exportar el proyecto.

2.4.4 Exportar el proyecto

Una vez se ha creado el proyecto, se puede exportar de varias formas. En este documento se demostrará el resultado para una exportación en PostgreSQL, MySQL y PNG.

2.4.4.1 PostgreSQL.

La intención es comprobar de manera gráfica con el programa pgModeler el resultado por parte de dbdiagram.io, comparando ambos resultados y ver si son idénticos.

Se utilizará la versión 9.5.4 de PostgreSQL para no tener problemas con pgModeler a la hora de conectar la base de datos al programa.

Se crea una base de datos nueva donde se pegará el código SQL importado desde dbDiagram.io y se ejecuta para crear todas las tablas.

En pgModeler se crea una conexión con la base de datos y se importan las tablas creadas en la base de datos. Es posible que se tenga que crear un usuario root en la base de datos, para esto ejecutar el siguiente comando:

```
ALTER USER nombre_usuario PASSWORD 'contraseña';
```

Crear un modelo nuevo en pgModeler e importar las tablas de la base de datos:

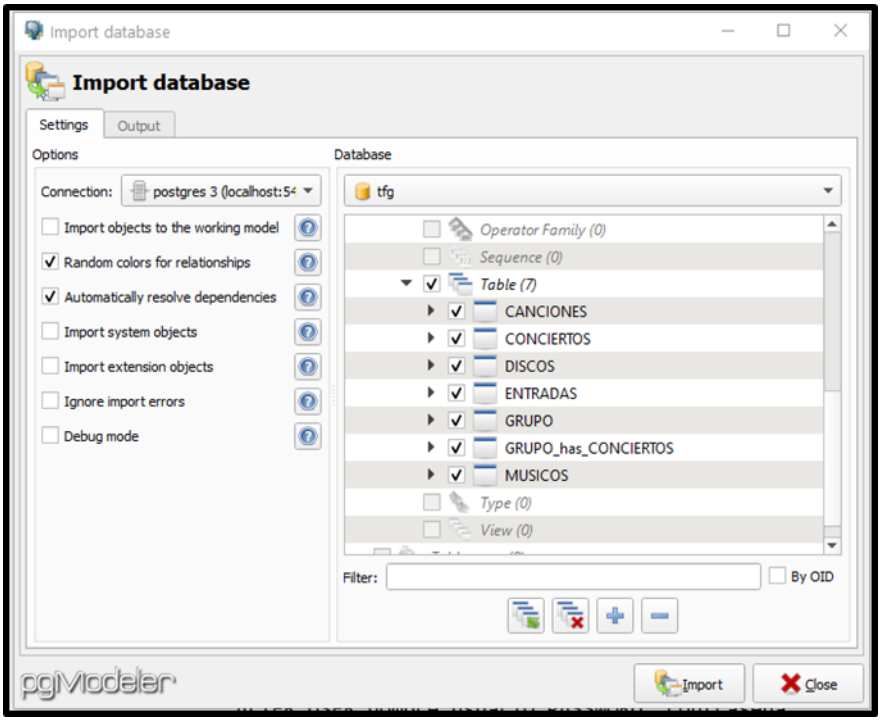


Figura 4. Importar base de datos en pgModeler.

Con todo esto, se mostrará el siguiente resultado (ver Figura 5):

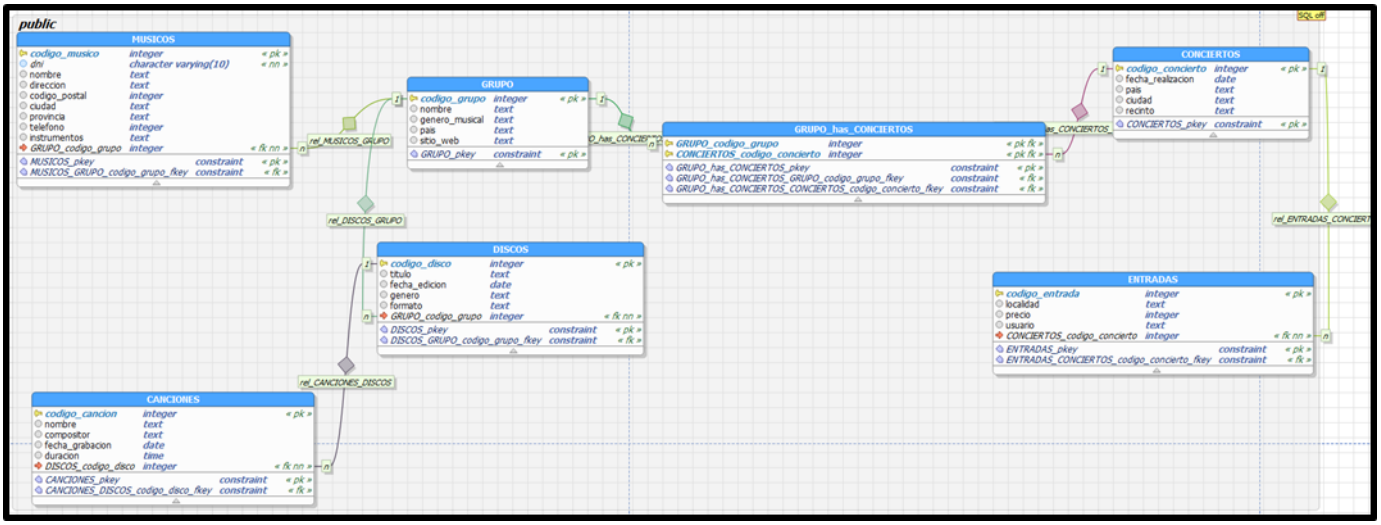


Figura 5. Resultado del diagrama exportado a partir de código SQL en pgModeler.

2.4.4.2 MySQL.

Cuando se exporta el código SQL desde dbDiagram.io, los nombres de las tablas y sus variables estarán rodeados por el símbolo de la tilde (`). Esto debe borrarse en todo el documento, en el caso que MySQL dé problemas a la hora de ejecutar la consulta, cambiar el símbolo por la comilla simple (‘).

En la versión de MySQL Workbench, existe un módulo para diseñar diagramas de entidad-relación, muy parecido a pgModeler. Desde la página principal, se accede a esta herramienta y se crea un nuevo modelo:

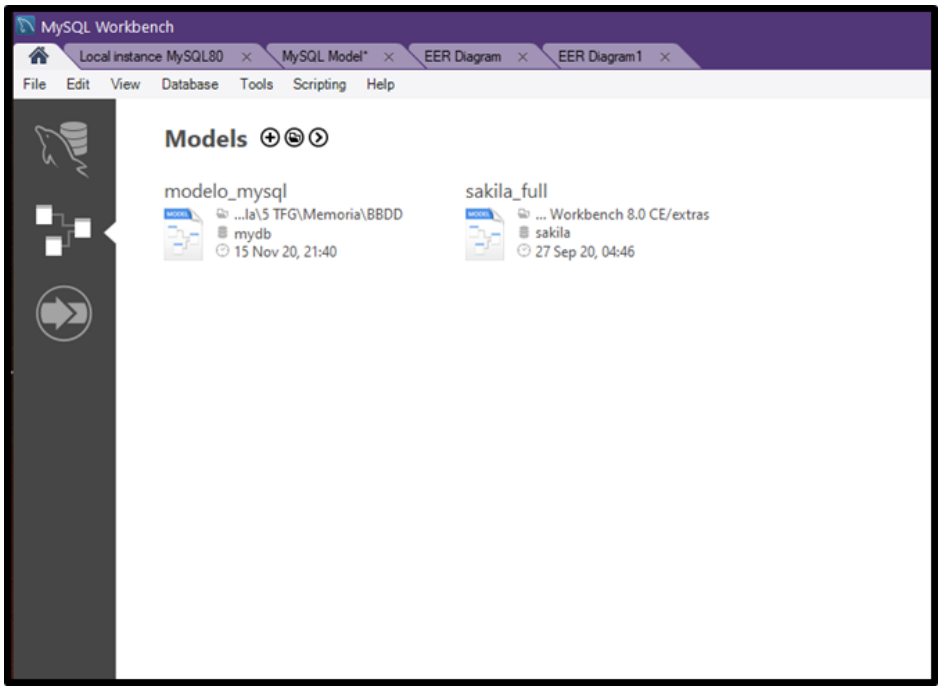


Figura 6. Pestaña de modelos en la ventana de MySQL Workbench.

Dentro, seleccionando “Add diagram” se mostrará una nueva ventana de diseño para empezar a dar forma al modelo. Existe un modo de reingeniería en la barra superior de la pantalla seleccionando File → Import… → Reverse Engineer MySQL Create Script. A continuación, Se mostrará la siguiente pantalla y habrá que seleccionar el código SQL exportado por dbDiagram.io y la opción “Place imported objects on a diagram”, de esta manera aparecerán las relaciones existentes entre las tablas.
Figura 7

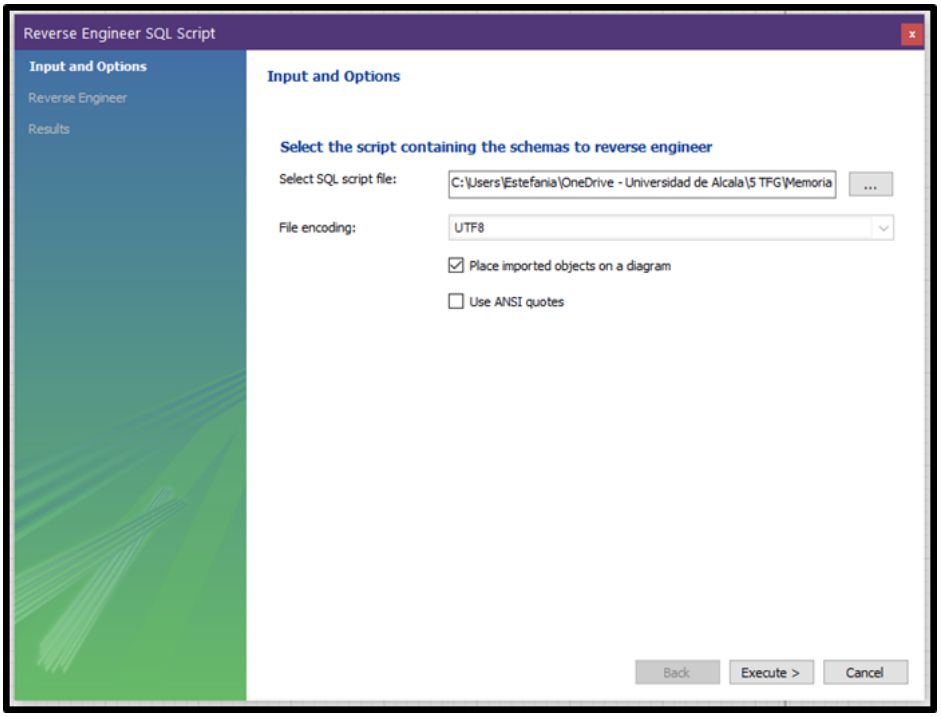


Figura 7. Importación del código en la opción de reingenería.

El resultado es idéntico a los dos modelos anteriores por dbDiagram y pgModeler.

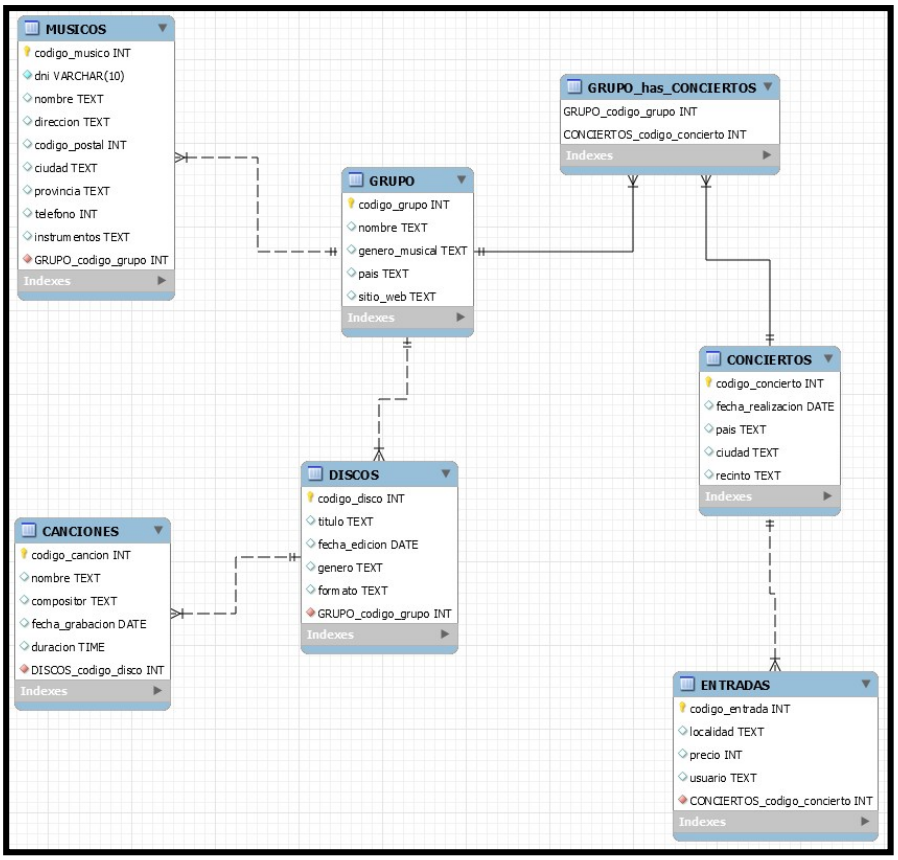


Figura 8. Resultado del diagrama exportado a partir de código SQL en MySQL Workbench.

2.4.5 Conclusiones de dbDiagram

DbDiagram.io es una herramienta muy sencilla e intuitiva para poder crear diagramas. Se agradece la documentación que aporta, ya que es bastante sencilla y clara. Sirve tanto para practicar código SQL como para editar diagramas ya existentes y exportarlos con facilidad a otros clientes. A pesar de ser una herramienta de pago, sus funciones gratuitas son suficientes para poder crear modelos entidad-relación sin problemas.

El único problema que se ha encontrado sobre esta herramienta es el soporte que ésta tiene. Las preguntas de los clientes que lo utilizan nunca obtienen respuesta.

Se ha tenido la experiencia de tener errores a la hora de importar código desde MySQL y también que la herramienta no sea compatible con ciertos aspectos de la nomenclatura del código que se utilizan en MySQL, por ejemplo:

- No se deben utilizar comillas en ningún caso, ni en el nombre de las tablas ni en los atributos.
- No es necesario mencionar el usuario que ha creado las tablas.

Además, las descripciones de los errores eran algo escuetas y se tuvo que buscar el problema por Internet sin éxito.

Se vio que otro usuario tenía el mismo problema (https://community.dbdiagram.io/t/expected-if-not-exists-or-valid-table-name-but-found/678) y no recibió nunca respuesta. De hecho, si se visitan otros temas en el foro de soporte, nadie recibe respuesta de nadie.

Después de llegar a comprender los errores de nomenclatura y resolverlos, se decidió enviar una respuesta al usuario que estaba teniendo el mismo problema. Por desgracia, después de un tiempo, se recibió este mensaje por parte del equipo de soporte, en el que se anuncia que han dado temporalmente de baja la cuenta que se creó para responder a este usuario sin ningún motivo:

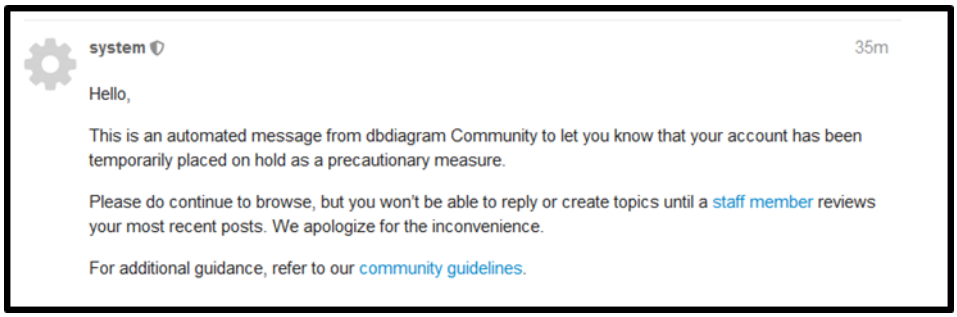


Figura 9. Mensaje de soporte de dbDiagram dando de baja la cuenta de su foro.

Se considera bastante importante un soporte adecuado para una herramienta como esta, donde los usuarios que tengan problemas de cualquier tipo puedan resolver sus problemas de manera rápida y pudiendo compartirlo con el resto de los usuarios para futuros errores similares.

A pesar de esto, se considera una herramienta adecuada para empezar con la materia en base de datos y para crear diagramas cuando no se tengan a mano programas como pgModeler o MySQL Workbench, ya que el hecho de poder observar a tiempo real el proceso de creación del modelo es muy cómodo para el usuario, tenga o no experiencia en este campo.

2.5 Diagrams.net

Esta página ofrece mucha variedad de modelos de muchos campos como software, ingeniería, UML, redes…, etc. y, en cada campo, ofrece variedad de diagramas disponibles. Estos diagramas se pueden guardar de manera local o en una nube en formato .dwaio para poder compartirlos o poder seguir editándolos.

Para crear un diagrama de entidad-relación, seleccionar el campo de Software y seleccionar el último diagrama. Si se coloca el puntero del ratón sobre cada imagen, se puede observar el nombre de cada diagrama:

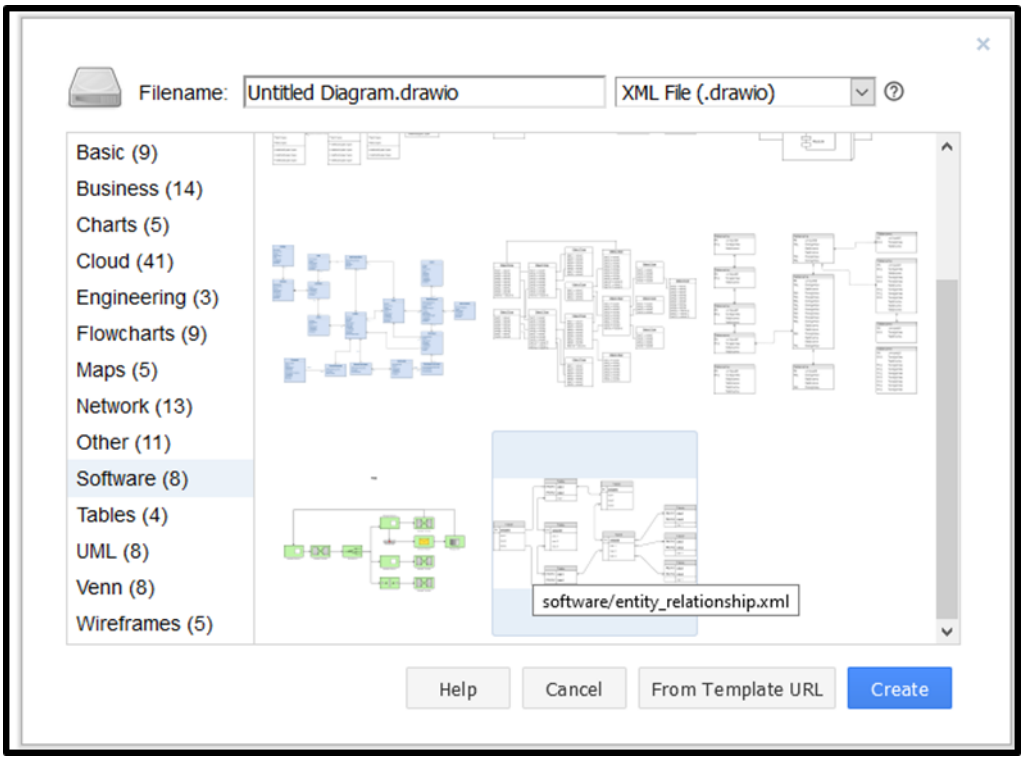


Figura 10. Creación de un diagrama entidad-relación en diagrams.net.

2.5.1 Pantalla principal

La pantalla principal se podría partir en 4 partes: superior (verde), izquierda (azul), central (morado) y derecha (rosa). (Ver Figura 11)

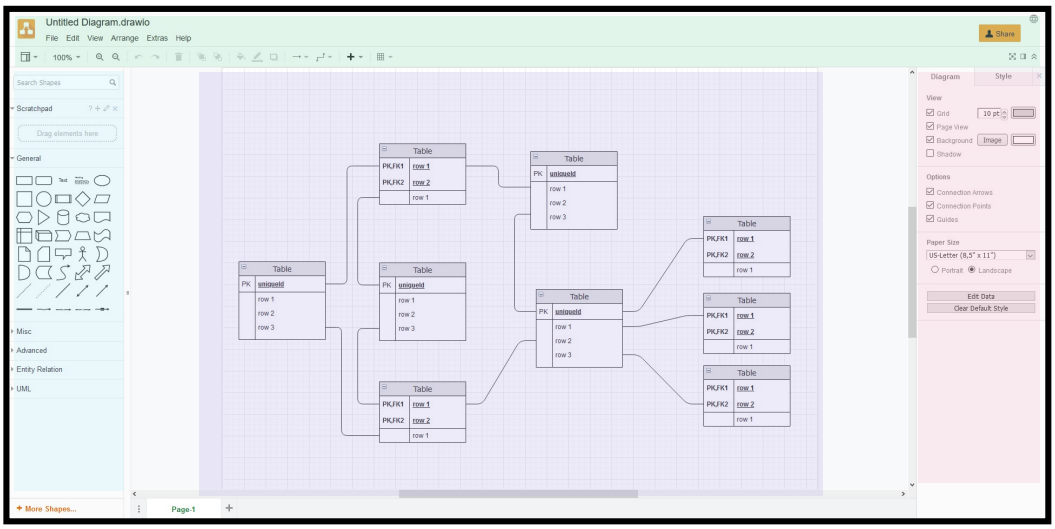


Figura 11. Pantalla principal en diagrams.net indicando sus diferentes partes.

En la parte superior se encuentra la barra de herramientas, donde se puede abrir un proyecto ya creado, crear uno nuevo o importar o exportar de opciones que ofrece la herramienta.

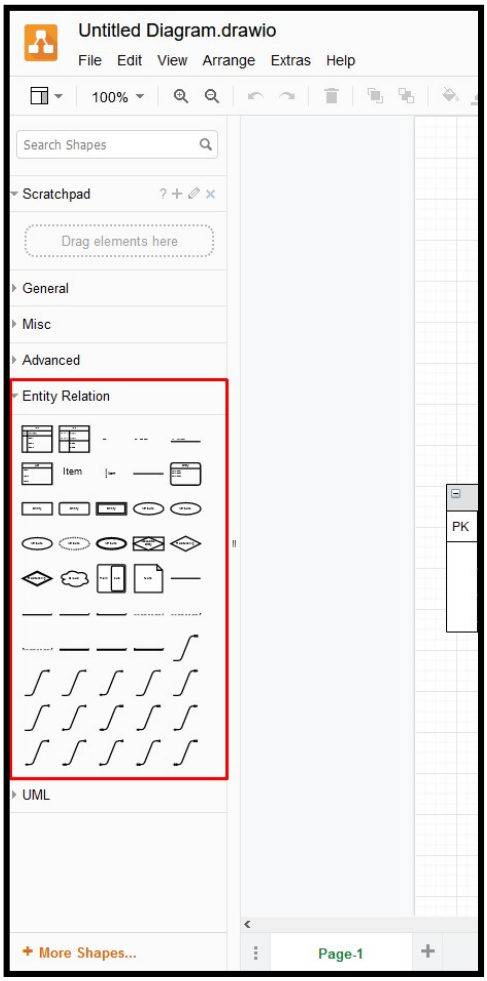
En la parte izquierda de la pantalla se encuentran los elementos que se pueden agregar al modelo que se va a crear. Se pueden añadir elementos de otros tipos de diagramas si es lo que se desea.

En la parte central es donde se editará el modelo en cuestión. Aquí aparecerán los elementos que lo forman.

En la parte de la derecha, se encuentra la parte del diseño del elemento que se tenga seleccionado en ese momento.

2.5.2 Crear tablas

Para crear tablas en un diagrama de entidad-relación, en la parte izquierda de la pantalla se encuentran los elementos de este diagrama:



Si se coloca el puntero del ratón sobre cada elemento, al lado saldrá una imagen del elemento indicando de qué tipo se trata. Para crear el elemento en la pantalla, se puede seleccionar dos veces y aparecerá en el centro de la pantalla o arrastrándolo al lugar que se desee.

2.5.3 Importar SQL.

Si se desea añadir tablas a partir de código SQL, primero se necesita instalar el plugin de SQL. Para esto, seleccionar la opción “Extras” de la barra superior y después “Plugins...”. Seleccionar el botón “Add” en el caso de que no esté ya incluido y en la lista que aparezca, seleccionar la opción “sql”. De esta manera, se añadirá el plugin una vez se actualice la página.

Para crear tablas a partir de SQL, seleccionar la opción “Arrange” en la parte superior y después “insert”. Entre todas las opciones, seleccionar “From SQL”. El código solo debe tratarse de la estructura de las tablas y se debe utilizar la siguiente nomenclatura [7]:

Figura 12. Herramientas relacionadas
con los modelos entidad-relación.

```
CREATE TABLE nombre (
Nombre_atributo tipo_variable NOT NULL
PRIMERY KEY (nombre_atributo_creado)
FOREING       KEY       nombre_atributo       REFERENCES       tabla2
(nombre_atributo_tabla2));
```

Un ejemplo del modelo que se está utilizando, creando las tablas Discos y Canciones:

```
CREATE TABLE Discos (
codigo_disco int
titulo text NOT NULL
fecha_grabacion date NOT NULL
genero text NOT NULL
formato text NOT NULL
PRIMARY KEY (codigo_disco)
);


CREATE TABLE Canciones (
codigo_cancion int
nombre text NOT NULL
compositor text NOT NULL
fecha_grabacion date NOT NULL
duracion time NOT NULL
codigo_cancion_d int
PRIMARY KEY (codigo_cancion),
FOREIGN KEY (codigo_cancion_d) REFERENCES Discos(codigo_disco)
);
```

Ambas aparecerán de esta manera:

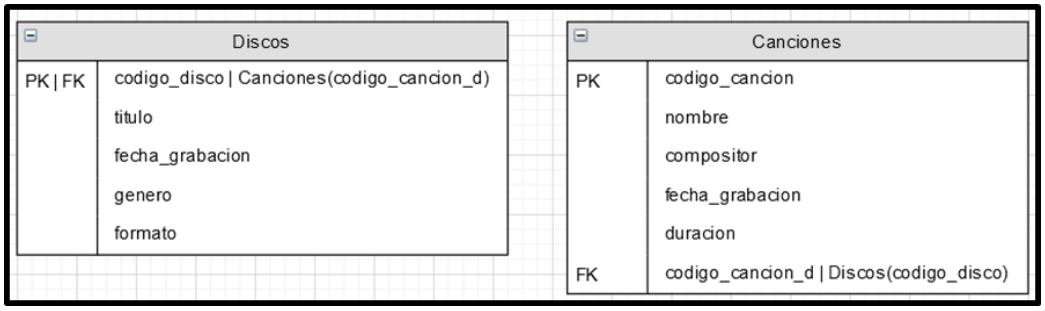


Figura 13. Tablas Discos y Canciones creadas a partir de su código SQL en diagrams.net.

Las relaciones no van a aparecer nunca cuando se crean las tablas, se deberán crear a mano.

2.5.4 Exportar proyecto.

Esta herramienta no permite exportar el diagrama a SQL y utilizarlo en algún cliente. En cambio, permite exportarlo en formato de imagen, PDF o en formatos web. Para ello, seleccionar “File” en la parte superior y seleccionar la opción “Export…”.

2.5.5 Conclusiones sobre diagrams.net

Esta herramienta solo sirve para crear diagramas de todo tipo y tiene una interfaz muy cómoda para el diseño. Ofrece muchos elementos y muchos tipos de modelos de diagramas orientados a cualquier producto informático. Permite importar datos de todo tipo y hace muy cómodo el hecho de poder guardarlo en diferentes plataformas y poder tener diversos diagramas en un mismo sitio.

A la hora de exportar el documento, sigue orientado al diagrama que se haya creado, sin poder sacar información tangible sobre el producto del diagrama, por tanto, no permite la ingeniería inversa y hace que la herramienta sea algo escasa a la hora de poder seguir explotando el diagrama.

Por tanto, esta herramienta solo está orientada al diseño de sus modelos de diagrama sin dar pie a poder utilizarlo en otros programas externos.

2.6 Otras herramientas

2.6.1 Con ingeniería inversa

2.6.1.1 SQL Database Modeler

Herramienta freemium donde pueden crearse modelos entidad-relación desde cero o a partir de código SQL. Si se utiliza con una cuenta gratis, solo se podrá crear y editar un proyecto a la vez, algo molesto si se tiene intención de llevar a cabo varios al mismo tiempo.

2.6.1.2 Visual diagrams

Contiene las mismas opciones que diagrams.net. La única diferencia es que ésta ofrece más tipos de diagramas. Su diseño y funcionamiento es el mismo.

2.6.2 Sin ingeniería inversa

2.6.2.1 DBDesigner

Herramienta freemium que permite crear diagramas entidad-relación y exportarlos a PDF o SQL para los siguientes clientes: MySQL, My SQL Server, SQLite, PostgreSQL y Oracle. Las cuentas creadas gratis pueden editar hasta dos modeles distintos dentro de esta herramienta.

2.6.2.2 DbDiffo

Al igual que la herramienta anterior, permite crear diagramas entidad-relación. Tiene un gestor de versiones y permite crear scripts de todo el diagrama o a partir de una versión concreta del proyecto. Este código puede ser exportado para varios clientes distintos: MySQL, PostgreSQL, IBM DB2, MS SQL Server y Oracle.

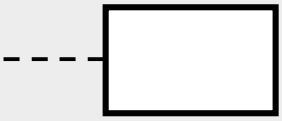
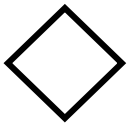
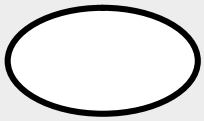
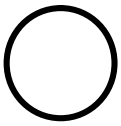
Capítulo 3. Diagramas de Flujo a partir de código

3.1 Diagramas de flujo.

Los diagramas de flujo son gráficos que representan el funcionamiento secuencial de un programa o proceso [8]. Existen varios tipos de estos diagramas [9]:

- Vertical. Las secuencias que representan el grafico se muestran de arriba abajo. Serán los diagramas que se mostrarán en este documento.
- Horizontal: en comparación con el anterior. Estos se muestran de izquierda a derecha.
- Panorámico: todo el proceso se muestra de una pieza.
- Formato arquitectónico: muestra el rol de una persona o forma sobre un aurea de trabajo que pueda existir en el proceso en cuestión.

Estos diagramas serán representados por simbología aprobada por la norma ISO 5807 [8] [10]:

FIGURA	NOMBRE FIGURA	DEFINICIÓN
	**Proceso**	Instrucciones del programa que convierte las entradas a salidas
	**Entradas / Salidas**	Información de entradas o salidas
	**Anotación**	Comentario o anotaciones que describen cualquier detalle del flujo
	**Decisión**	Condicionales que crean diferentes decisiones en el programa
	**Comienzo / Final**	Indica el comienzo o final del programa
	**Conector**	Sigue o acaba en otra parte del diagrama (para diagramas grandes)
	**Conector**	Sigue o acaba en otra parte del diagrama que se encuentra lejano (por ejemplo, en otra página)

	**Proceso Predefinido**	Subrutinas o procedimientos existentes en el programa.
	**Flecha**	Une los elementos del diagrama y muestra el sentido de éste.

Figura 14. Simbología de los diagramas de flujo según la norma ISO 5807.

Gracias a este diagrama, se resume de manera visual cualquier funcionamiento de un programa o proceso que se haya creado, favoreciendo de esta manera la comprensión para nuevos usuarios o personas sin conocimiento de programación, en el caso de diagramas de flujo sobre proyecto software. También ayuda a darse cuenta de posibles cambios o errores existentes en el proyecto.

3.2 Ejemplo para implementar.

El flujo se creará a partir de un script escrito en Java y C++. Originalmente está escrito en Java y su fuente es de esta página web: https://www.geeksforgeeks.org/java-program-to-implement-the-monoalphabetic-cypher/. Este código de ejemplo tratará de encriptar y desencriptar una cadena de caracteres. Para la creación del diagrama de flujo solo se utilizará la función que encripta el texto.

Se encontrará el script tanto en Java como en C++ en el repositorio de GitHub en el directorio del capítulo 4: https://github.com/docestavivo/TFG. El script en C++ solo contendrá la función encryptedString.

Se recorrerá la longitud del string s (previamente convertidas todas las letras en minúsculas) y se comprobará cada letra que se detecte (solo letras) a partir de un array con las letras del abecedario y serán sustituidas después por letras que correspondan en otro array con las letras del código secreto. A continuación, se muestra los arrays:

```
public static char normalChar[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z' };
```

```
public static char codedChar[] = { 'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P', 'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'Z', 'X', 'C', 'V', 'B', 'N', 'M' };
```

3.3 Code2Flow.

Code2Flow se trata de una web para crear diagramas de flujo a partir de su propio pseudocódigo o a partir de código escritos en Java, C o C++. La herramienta es freemium y para cuentas gratuitas permite crear hasta 50 nodos por flujo. Los flujos puedes exportarse de diferentes maneras posteriormente:

- Como link para compartir con otras personas que no necesariamente deben tener una cuenta en code2Flow. Pueden verlo como el proyecto en sí, en png, en svg y en pdf de manera online.
- El diagrama también se puede descargar en formato png, svg y pdf.

En ambos casos, en los formatos externos puede incluirse el código que representa el diagrama.

3.3.1 Pantalla principal.

La pantalla principal cuando se crea un nuevo diagrama es la siguiente:

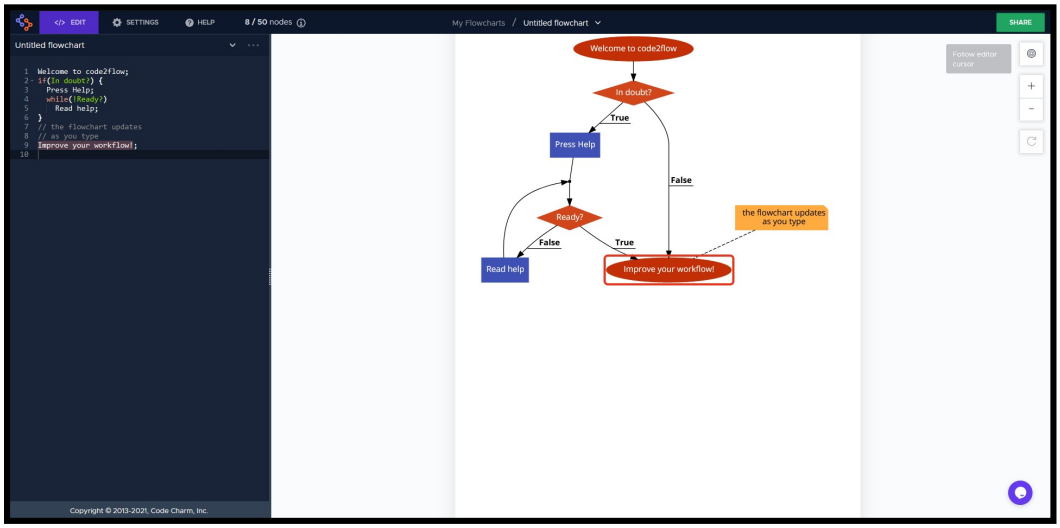
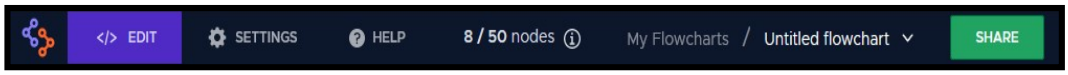


Figura 15. Pantalla principal de code2flow.

En la parte superior de pantalla se encuentran las siguientes opciones:



1 2 3 4 5 6 8

Figura 16. barra superior de la pantalla principal de code2flow.

Cuando se selecciona uno de estos botones, éste se rellenará de color morado.

1. **Icono de la web:** Si se selecciona este botón, volverá al repositorio donde se encuentren los proyectos creados por el usuario.
2. **Edit:** En este caso, si se selecciona cerrará el panel que muestra el script del diagrama. En el caso de encontrarse en otra pestaña, seleccionando esta se volverá a aparecer el apartado del código.
3. **Settings:** Este botón muestra las diferentes configuraciones que permite la web para el diseño del código. Se mostrará en la parte izquierda de la pantalla:
    - Content settings: En este apartado se configura el texto que corresponda con los condicionales en el caso de ser verdadero o falso. Permite activar o desactivar un optimizador del diseño del diagrama en el caso que haya nodos que contengan el mismo texto. Y, por último, activar o no decorador a textos que se encuentren en las líneas que unen los nodos.
    - Rendering settings: Estas opciones corresponden con aspectos sobre el diagrama: el tema de colores del diagrama, la escala del diagrama, dirección

de la expansión del diagrama, tamaño de la fuente, y si se desea que se muestre de manera compacta.
    - Por último, se encuentra un botón que reestablece todas las opciones actuales a las predeterminadas por la web.
4. **Help:** Muestra la documentación necesaria para conocer el pseudocódigo con el que se crea el diagrama.
5. **Número de nodos:** muestra el número de nodos usados por el momento y su límite. Para cuentas gratuitas habrá máximo 50 nodos a usar. Para cuentas de pago, hasta 200 nodos.
6. **Proyectos:** En esta parte, se muestra el directorio donde se encuentra el proyecto actual y si se selecciona, se podrán ver diferentes opciones que se pueden aplicar al proyecto:
    - Share: Para compartir un enlace del diagrama mostrando o no el código.
    - Download: para descargar el diagrama en formato png, svg o pdf mostrando o no el código.
    - Embed: para compartir el diagrama a través de un enlace en formato png, svg o pdf.
    - Version history: esta opción aún no se encuentra desarrollada.
    - Rename flowchart: permite renombrar el diagrama.
    - Clone: permite clonar el diagrama para crear otro similar y ahorrar tiempo de esta manera.
    - Move to Project: Mover el diagrama a otra carpeta creada por el usuario.
    - Delete: Borrar el diagrama.
7. **Share:** Permite compartir o exportar el diagrama mostrando o no el código.

La pantalla anterior a esta se trata del repositorio de la web donde se encuentran todos los proyectos que se han creado hasta ese momento:

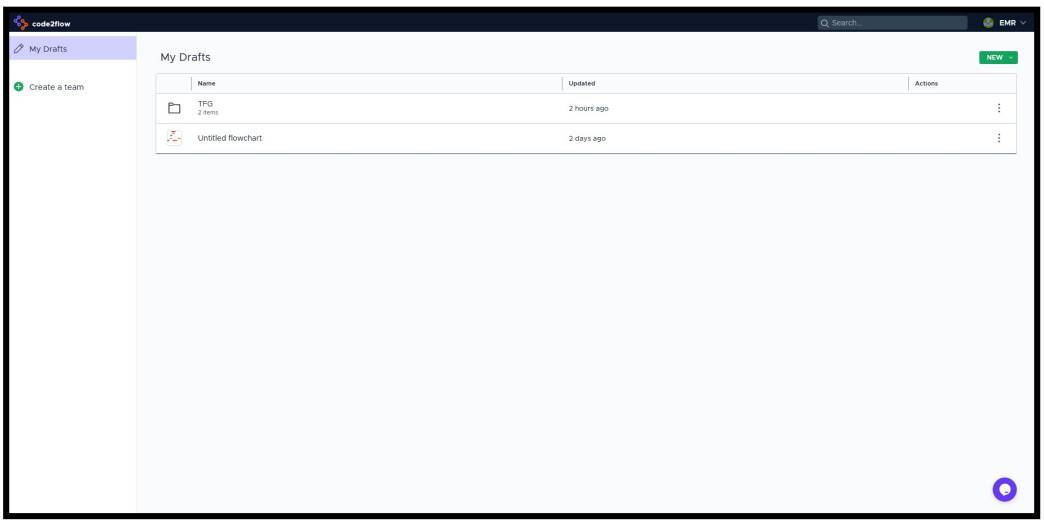


Figura 17. Repositorio de code2flow.

3.3.2 Crear un diagrama.

3.3.2.1 Flujo en Java.

Flujo creado a partir de código en Java a partir de la funciona encriptadora del script: https://app.code2flow.com/0DZ4NsX3HrfP.

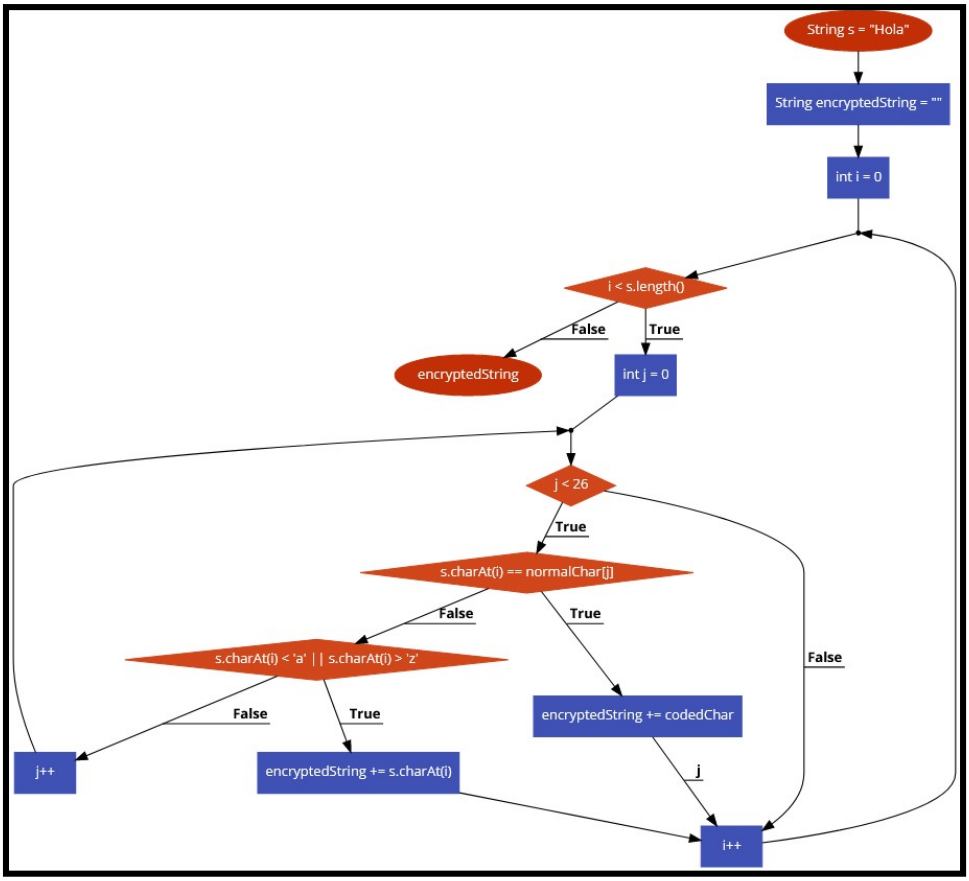


Figura 18. Diagrama de flujo a partir de un script en Java.

3.3.2.2 Flujo en C++.

Flujo creado a partir de código en C++ a partir de la funciona encriptadora del script: https://app.code2flow.com/qv79bZfeBDPh.

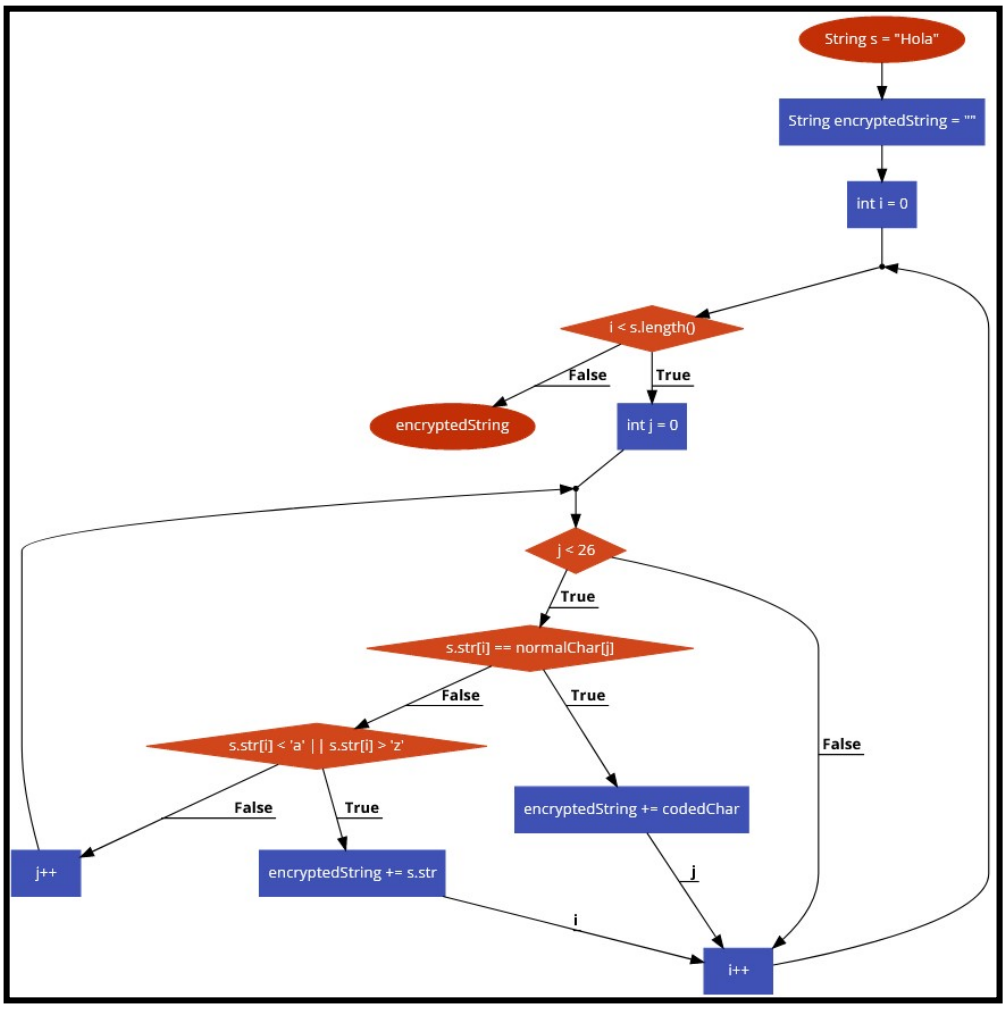


Figura 19. Diagrama de flujo creado a partir de codigo en C++.

Se puede observar que a pesar de que son dos códigos ligeramente distintos, el flujo no cambia en cuanto a la estructura.

Los aspectos que hay que destacar:

- Los condicionales con doble condición como es el caso del segundo IF no se separa por partes.
- Ayuda que se indique qué variable pasa a la siguiente instrucción de un condicional como en el caso de los contadores después de los IF o los valores de las condiciones.
- No hay problemas de nomenclatura del código a la hora de pegar un script desarrollado en Java o C++.
- Acepta librerías a pesar de no indicarlas, por ejemplo, la librería <string> en C++.

- Permite comentarios y los muestra con una línea discontinua sobre el nodo de la instrucción siguiente a donde se encuentre (también se contarán como nodos):

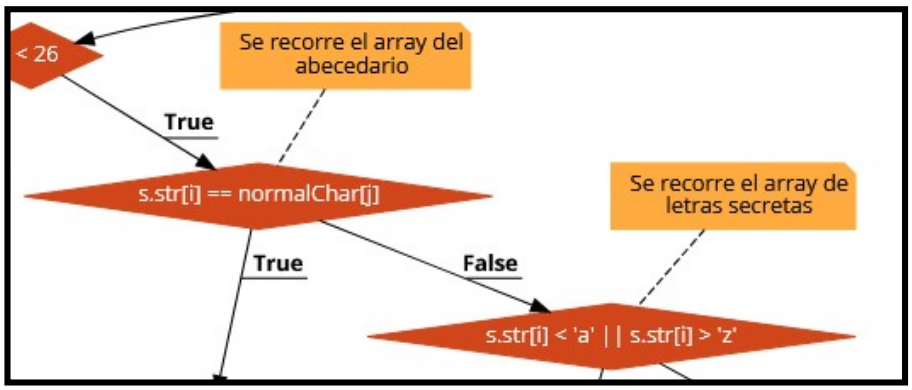


Figura 20. Comentarios en un diagrama de flujo.

- Se pueden mezclar lenguajes de código sin recibir avisos o error por parte de code2flow.

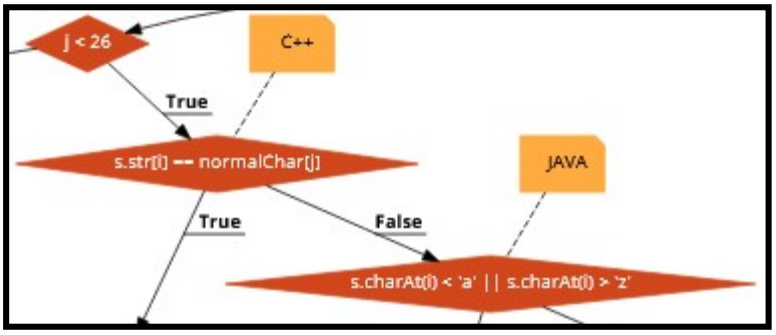


Figura 21. IF de la izquierda escrito en C++ e IF derecho en Java.

```
//C++
if (s.str[i] == normalChar[j]){
encryptedString += codedChar[j];
break;
}

//JAVA
if (s.charAt(i) < 'a' || s.charAt(i) > 'z'){
    encryptedString += s.charAt(i);
    break;
    }
```

Figura 22. Desarrollo de los IF en diferentes lenaguajes: C++ y Java.

- En el caso de desarrollo de condicionales de tipo while, se muestran ligeramente parecidos a los for:

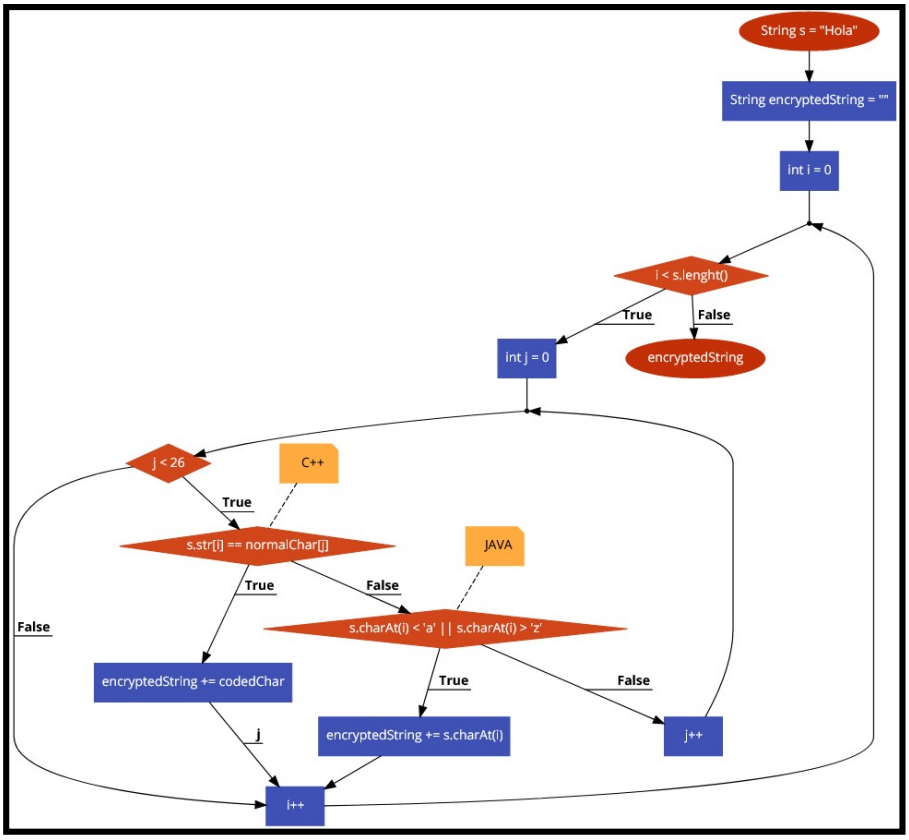


Figura 23. Flujo que sustituye el primer for por un while.

```java
String s = "Hola";
String encryptedString = "";
int i = 0;

while (i < s.lenght()){
  for (int j = 0; j < 26; j++){

    //C++
    if (s.str[i] == normalChar[j]){
    encryptedString += codedChar[j];
    break;
    }

    //JAVA
    if (s.charAt(i) < 'a' || s.charAt(i) > 'z'){
        encryptedString += s.charAt(i);
        break;
        }
  }
  i++
}

return encryptedString;
```

Figura 24. Desarroollo del flujo con el primer for sustituido por un while.

3.3.3 Conclusiones.

Se trata de una herramienta muy intuitiva y que, a pesar de tener limitaciones para usuarios gratuitos, son suficientes para proyectos pequeños.

Se considera importante que permita poder crear diagramas no solo a partir de su pseudocódigo si no a partir de códigos desarrollados en Java o C/C++, ya que su propio pseudocódigo está inspirado en estos lenguajes de programación. Esto agiliza mucho el hecho de querer recuperar información de un script del que se haya perdido parte de su documentación y se quiera, por ejemplo, conocer el comportamiento de ese programa. Tal vez el modo gratuito limite mucho esta recuperación para proyectos muy grandes, considero que solo puede usarse para programas pequeños o para conocer el funcionamiento de ciertas funciones que pueda contener un programa y no sobrepasen este límite.

El único inconveniente que se le encuentra es que permita mezclar lenguajes sin mostrar ningún tipo de aviso o error, como ya se ha mostrado en el apartado 2.2, ya que, a la hora de exportar el diagrama, puede llevar a errores futuros, ya que tanto Java como C/C++ son lenguajes muy parecidos y pueden ser fácilmente ignorados estos errores de nomenclatura hasta que no se pase un compilador sobre el código en cuestión.

3.4 Code visual to flowchart.

Se trata de un programa compatible para Windows que crea diagramas de flujo a partir de una gran lista de lenguajes de programación. En esta versión DEMO que se mostrará a continuación solo se podrán exportar los scripts que forman los flujos en formato rtf o HTML. En la versión de pago se permite exportar los flujos para Word, Visio, PowerPoint, Excel o imagen en formato PNG o BMP. También se limitarán los flujos a un máximo de tres niveles.

Los flujos que se muestran en el documento se muestran a partir de capturas de pantalla.

3.4.1 Pantalla principal.

La pantalla principal del programa se muestra de esta manera:

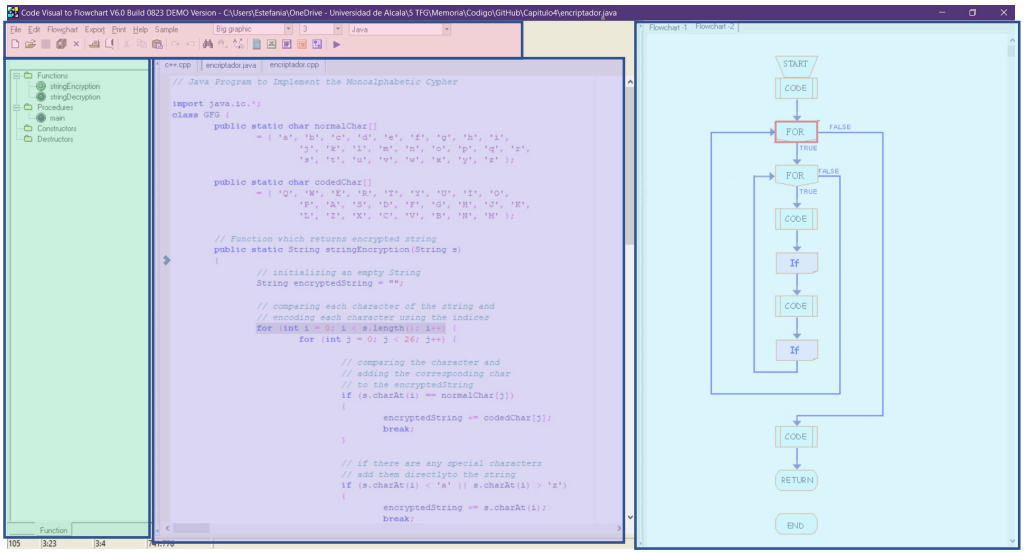


Figura 25. Pantalla principal de Code Visual to Flowchart.

Se partirá la pantalla en cuatro partes:

- En la parte superior (rosa), se encuentran las herramientas del programa. Los más importante a destacar son las opciones:
  - Export: Permite exportar los flujos en diferentes formatos:

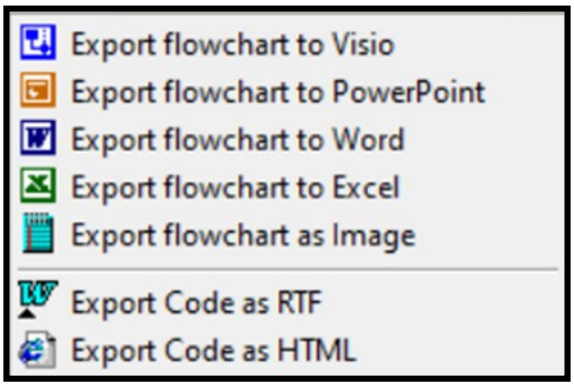


Figura 26. Diferentes formatos de exportación en Code Visual to Flowchart.

  - Sample: muestra un ejemplo en los lenguajes que muestra.

  - Configuraciones de los flujos: La primera opción permite lo siguiente:
    - All Code: Crea un flujo sobre todo el código, de principio a fin (no disponible en la versión DEMO).
    - Without comment: Flujo sin mostrar los comentarios existentes en el script.
    - Only comment: Solo muestra si se trata de un bucle, condicional o líneas de instrucciones y los comentarios del script.
    - Small Graphic: muestra un diagrama muy simplificado con siglas de cada elemento de los nodos.
    - Big Graphic: muestra un diagrama que resume el flujo del programa sin mostrar información sobre el código.

    La siguiente opción permite mostrar los diagramas por niveles, en la versión demo solo se permite de uno a tres niveles.

    La última opción es para indicar el lenguaje del script del que se desea crear un diagrama.

- En la parte izquierda de la pantalla (verde), se muestran los elementos disponibles del código abierto en ese momento, como las clases o las funciones del script.

- En la parte central (morado) de la pantalla se encuentra el script. En la parte superior se muestran todos los scripts abiertos.

- En la parte derecha (azul) se muestran los flujos. En la parte superior se encuentran las pestañas 1 y 2, que muestran el diagrama con detalles o sin ellos respectivamente.

3.4.2 Crear un diagrama.

Para crear un diagrama, se abrirán los scripts de la función de encriptar un texto que se utilizó en la herramienta code2Flow. Se abrirá tanto en Java como en C++.

Es importante que el script contenga la clase que lo crea, de otra manera, el programa no creará el diagrama.

3.4.2.1 Diagrama con Java.

A diferencia con el script de C++, este contiene todo el programa completo, tanto la función que encripta como la que desencripta. Solo se creará el flujo sobre la función encryptedString, para ello, se selecciona la línea en la que empieza la función, en este caso la línea 16 y con el botón derecho del ratón se selecciona la opción "Mark begin line of code", de esta manera se indica en qué línea del código empieza el flujo. De la misma manera, para indicar el final, se selecciona la opción "Mark end line of code".

Otra manera de crear el flujo es dando dos veces a la línea de código y el programa creará un flujo hasta el corchete final de esa línea de código seleccionada, pero puede darse el caso de que no muestre lo que se desea.

El flujo se mostrará de la siguiente manera (ver Figura 27. Diagrama de flujo de la función stringEncryption en Java en Code Visual to Flowchart.Figura 27):

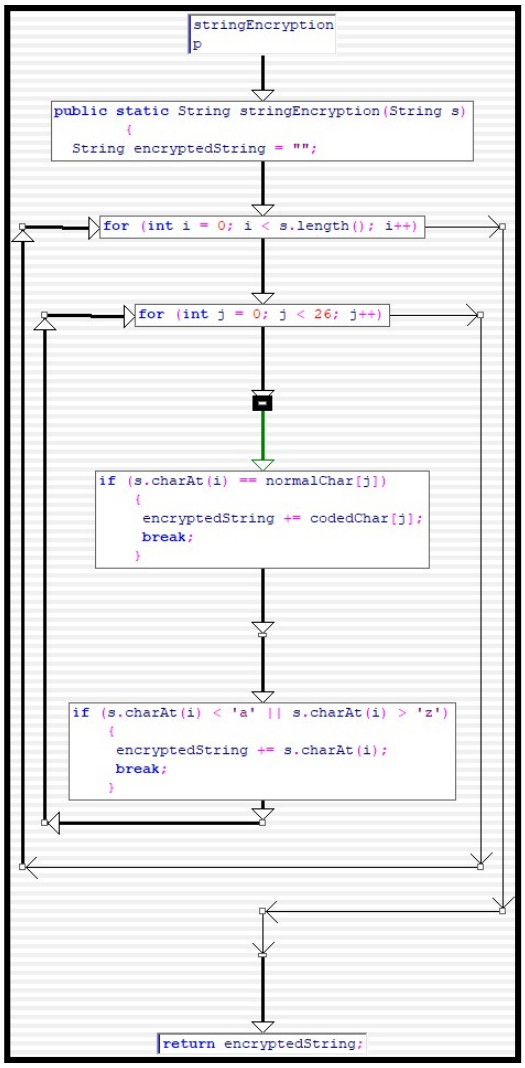


Figura 27. Diagrama de flujo de la función stringEncryption en Java en Code Visual to Flowchart.

3.4.2.2 Diagrama en C++.

El script de este lenguaje solo tratará la función encriptadora de texto. En este caso, para crear el diagrama correctamente, se debe empezar el flujo a partir de la línea 14 y acabar en la línea 33 de la misma manera que se ha hecho en el apartado anterior. De esta manera, se mostrará de la siguiente manera:

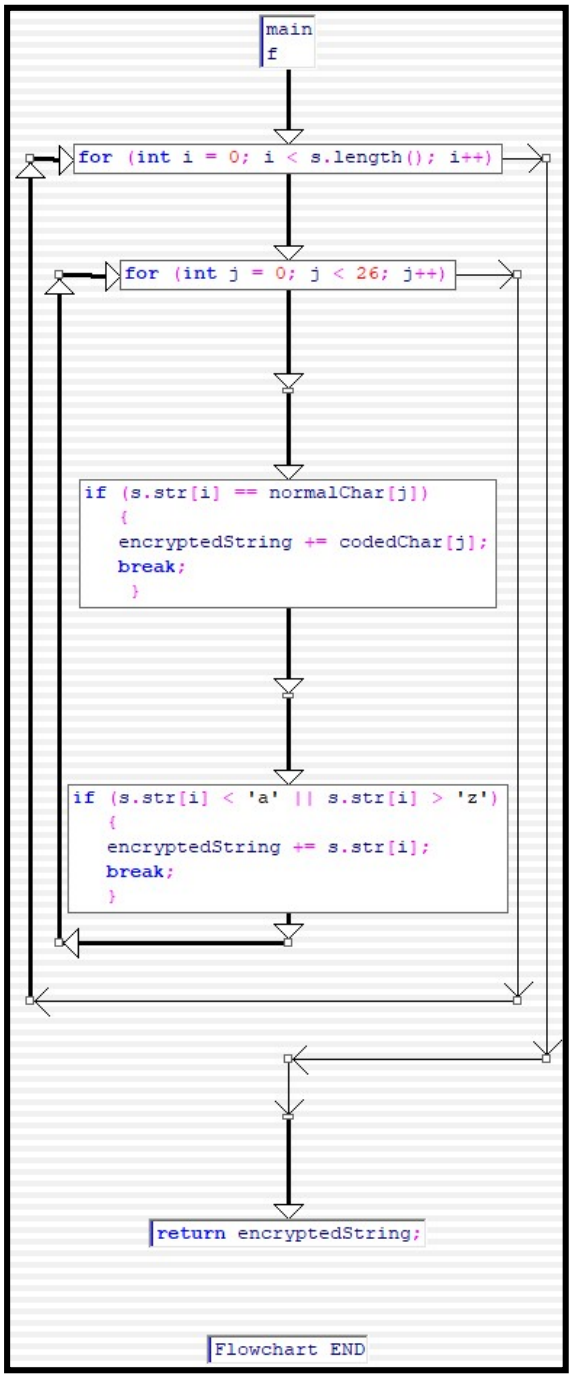


Figura 28. Diagrama de flujo de la función stringEncryption en C++ en Code Visual to Flowchart.

3.4.3 Diagramas grandes.

El programa también dedica un apartado a crear un diagrama llamado “Big Graphic Diagram” que presenta el flujo del programa de manera resumida sin detalles. Estos diagramas también pueden presentarte con diferentes niveles.

Se encuentra en la parte superior de la ventana donde se muestran los diagramas, seleccionando el botón “Flowchart – 2”, de esta manera, los diagramas anteriores se mostrarían así (ambos son iguales):

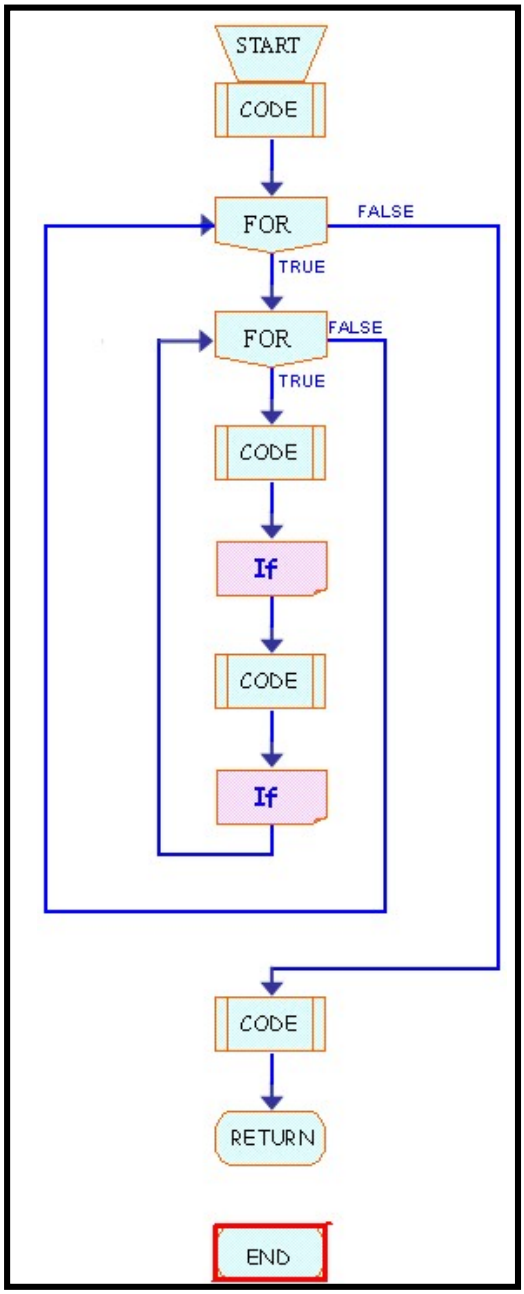


Figura 29 . Diagrama grande de la función stringEncrypted.

3.4.3 Conclusiones.

Los principales defectos que presenta el programa es que la versión DEMO limita mucho el desarrollo de posibles proyectos grandes que requieran muchos niveles en sus flujos. También hay que mencionar que la limitación de no poder exportar los flujos a

imagen u otros formatos presenta muchos obstáculos a la hora de querer diseñar documentación futura sobre el proyecto.

Se considera este programa adecuado para proyectos pequeños donde se quiera consultar los procedimientos de un programa de manera individual. Para el ámbito de la docencia donde los alumnos puedan observar el funcionamiento de un programa de manera visual, este programa es apto. Por último, queda recalcar que el programa permite crear flujos de una gran cantidad de lenguajes de programación diferentes, facilitando poder utilizar el programa para múltiples proyectos diferentes.

3.5 Otras herramientas.

Otras herramientas online disponibles para crear flujos a mano son las siguientes:

- Lucidchart (https://www.lucidchart.com): permite crear diagramas de flujo gracias a sus múltiples herramientas de diseño.
- Smartdraw (https://cloud.smartdraw.com): igual que la anterior.
- Flowchart.js (https://github.com/adrai/flowchart.js): permite crear flujos a partir de comando creados por el creador en JavaScript. Toda la información está disponible en GitHub.
- Chartmage (http://chartmage.com): Permite crear diagramas de flujo o diagramas secuenciales a partir de código. Es más limitado que code2flow.

Capítulo 4. Medición del Software

4.1 Introducción a la medición del software.

Para conocer cualquier cosa tangible o no, como puede ser el software, es necesario poder expresarlo de forma numérica gracias a las mediciones disponibles en cada caso. De esta manera, se podrá conocer con más detalle y precisión aquellos objetivos que se desea realizar como pueden ser el control posibles mejoras o mejor compresión del producto [11].

En el caso de mediciones del software se mencionarán las medidas que se utilizarán en este capítulo. Estas medidas sirven para cualquier producto y expresan diferentes atributos, por ejemplo, las líneas de código (LOC) o Halstead expresan longitud y la complejidad ciclomática sobre la estructura del código.

4.1.1 Complejidad ciclomática.

Se trata de una medida útil para pruebas de caja blanca e identificar la complejidad que presentan las líneas de código de un programa. En primer lugar, se identifica el flujo del programa. Por cada sentencia existirá a un nodo. Los bucles o condicionales formarán ramificaciones (aristas) que aumentarán la complejidad del flujo.

Una vez identificado el flujo, la complejidad puede ser calculada de 3 maneras distintas que darán el mismo resultado [11]:

V(G) = aristas – nodos + 2
V(G) = nodos predicado + 1
V(G) = nº de regiones cerradas

Los nodos predicado se tratan de nodos con más de una salida.

El resultado serán los caminos independientes del flujo y por cada resultado existirá un caso de prueba [12]:

Complejidad calculada	Resultado
1 - 10	Código estructurado y correcto *Testabilidad* alta El coste y el esfuerzo de mantenimiento son bajos.
10 – 20	Código complejo Testabilidad media El coste y el esfuerzo de mantenimiento son medios.
20 – 40	Código muy complejo Testabilidad baja El coste y esfuerzo de mantenimiento son altos
>40	No se puede probar

El coste y el esfuerzo son excesivos

Tabla 1. Clasificación de las complejidades calculadas.

Con esto se puede concluir que cuanto mayor sea el número de decisiones en el programa la complejidad también será mayor y por tanto la aparición de errores también incrementará incrementando de esta manera el futuro tiempo de mantenimiento y la resolución de posibles problemas.

4.1.2 Halstead.

Este caso cuenta todos los elementos o también llamados token que se presentan en el código. Todas las variables y sus valores se consideran operadores. Los tipos de variables y elementos del código son operados. Los operandos y operadores únicos se representarán como n_1 y n_2 respectivamente, el número de apariciones de cada uno de ellos se representará como N_1 y N_2.

Se cuenta cada aparición de estos tokens por separado y se calculan los siguientes atributos [11][13]:

Longitud (N) = $N_1 + N_2$
Vocabulario (n) = $n_1 + n_2$
Volumen (V) = $N \cdot log_2(n)$

Las siguientes fórmulas no se consideran tan aceptadas como las tres anteriores, pero cabe mencionar su existencia:

Dificultad (D) = $n_1/2 \cdot N_2/n_2$
Esfuerzo (E) = $D \cdot V$
Tiempo de programación en seg = $E/18$

Estos resultados deben ir acompañado de otras medidas como las que se mencionan en este capítulo ya que estos resultados resultan ambiguos y en el cálculo de las demás ediciones.

4.1.3 Número de líneas de código.

Es la medida más simple a la hora de calcular ya que consiste en contar las líneas existentes en el programa.

- Se le puede poner restricciones, por ejemplo:
- Contar las líneas con o sin comentarios.
- Ignorar o no las líneas blancas.
- Tomar en cuenta o no las declaraciones de variables o las cabeceras.
- Contar como una de las sentencias que ocupan dos líneas o más.

4.2 Código para pruebas.

En este capítulo se utilizará un algoritmo llamado Flood Fill disponible en el siguiente link: https://www.sanfoundry.com/java-program-flood-fill-algorithm/. Está desarrollado en Java.

Consiste en determinar el área conectada a un nodo en un array o matriz multidimensional. Se utiliza, por ejemplo, en programas de dibujo para rellenar líneas de un mismo color (el botón en

forma de cubo de pintura). También se utiliza en el juego del Buscaminas para despejar una zona grande del tablero.

Las 'P' se tratan del nodo en cuestión, las 'W' del relleno de este nodo 'P' y las 'O' de obstáculos. Se trata de convertir todas las 'P' en 'W', ignorando las 'O'.
El código también se encontrará disponible en el repositorio GitHub del proyecto: https://github.com/docestavivo/TFG/tree/master/Capitulo5.

4.3 Uso de Lizard.

Lizard es una herramienta para medir la complejidad ciclomática de las funciones de un programa en diferentes lenguajes como C/C++, Java, C#, Python, Lua… entre otros muchos. En concreto cuenta con las siguientes mediciones de cualquier script:

- NLOC: no cuenta comentarios y líneas blancas. Las líneas partidas se cuentan como nuevas líneas.
- McCabe: Complejidad ciclomática.
- Halstead: token de las funciones.

También es capaz de detectar repeticiones en el código.

Esta herramienta puede utilizarse online, como se mostrará en este documento, y también puede instalarse como explica el autor en el repositorio GitHub donde se encuentra alojada: https://github.com/terryyin/lizard.

4.3.1 Pruebas en Java.

En la herramienta online de Lizard se ha pegado el script completo en su formulario y se ha mostrado el siguiente resultado:

Code analyzed successfully.

File Type .java | Token Count 442 | NLOC 45

Function Name	NLOC	Complexity	Token #	Parameter #
FloodFill::fillGrid	10	2	90	
FloodFill::display	8	3	86	
FloodFill::main	23	5	240	

Figura 30. Resultados de Lizard para el script en Java.

Los resultados consisten en:

- El tipo de archivo del código. En este caso es un .java.
- Los elementos o token totales en el programa. En el caso de este script hay 442 elementos.
- El número de líneas de código sin comentarios ni líneas blancas. Se han contabilizado 45 líneas.
- Después mostrará las líneas de código, la complejidad y los tokens de cada método encontrado en el script.

Se mostrarán a continuación las justificaciones de estas medidas:

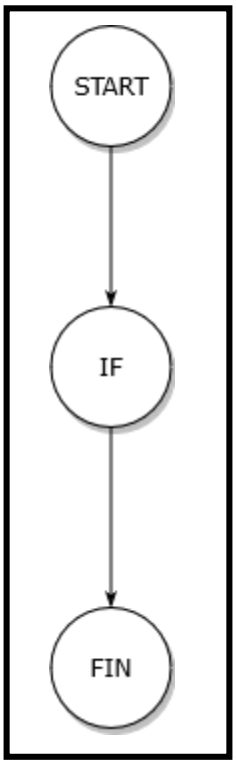
4.3.1 Función fillGrid().

4.3.1.1 Número de líneas de código.

Como ya se ha mencionado, Lizard no cuenta las líneas con comentarios y líneas blancas. Por tanto, para esta función se han contabilizado 10 líneas:

- IF: 8 líneas.
- El resto: 2 líneas.

4.3.1.2 Complejidad ciclomática.



4.3.1.3 Halstead. Tokens de la función.

Para esta función, se contabilizan 90 tokens entre los operadores y operandos. Destacar que no se cuenta el encabezado de la función, así como el tipo de acceso (public, private o protect), modificador (static o final) y el tipo de función (int, float, char, String…), solo se cuentan los parámetros de la función. Los tokens son los siguientes:

Operandos	Nº	Operadores	Nº
fillGrid	5	()	14
arr	8	char	1
r	7	[]	12
c	7	int	2
‘P’	1	,	10
‘W’	1	{}	4
display	1	if	1
1	4	==	1
		=	1
		;	6
		+	2
		-	2
TOTAL	**34**	**TOTAL**	**56**

Tabla 2. Tokens en la función fillGrid().

Longitud (N) = $N_1 + N_2 = 8 + 12 = 20$
Vocabulario (n) = $n_1 + n_2 = 34 + 56 = 90$
Volumen (V) = $N \cdot \log_2(n) = 20 \cdot \log_2(90) = 129{,}83$

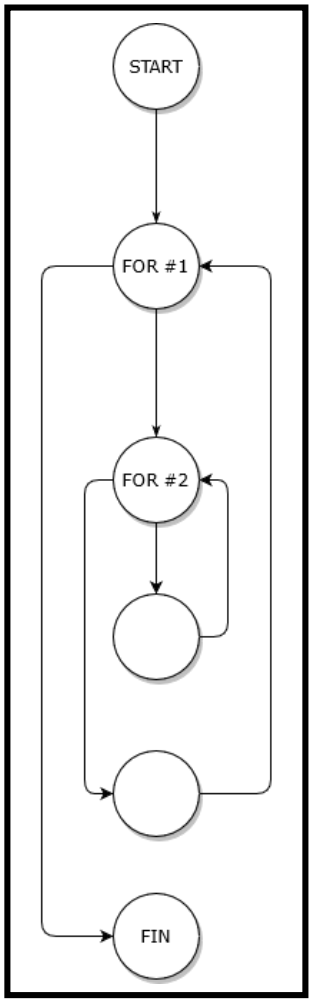
4.3.2 Función display().

4.3.2.1 Número de líneas de código.

Para esta función se han contabilizado 8 líneas:

- Segundo FOR: 2 líneas.
- Primer FOR: 3 líneas.
- Resto: 3 líneas.

4.3.2.2 Complejidad ciclomática.



4.3.2.3 Halstead. Tokens de la función.

Para esta función, se contabilizan 86 tokens entre los operadores y operandos. Los tokens son los siguientes:

Operandos	Nº	Operadores	Nº
display	1	()	12
arr	4	char	1
"\nGrid : "	1	[]	10
i	5	{}	4
1	4	System	3
" "	1	.	8
j	4	out	3
		println	2
		print	1
		;	7
		for	2
		int	2
		=	2
		<	2
		length	2
		-	2
		++	2
		+	1
TOTAL	**20**	**TOTAL**	**66**

Tabla 3. Tokens de la función display().

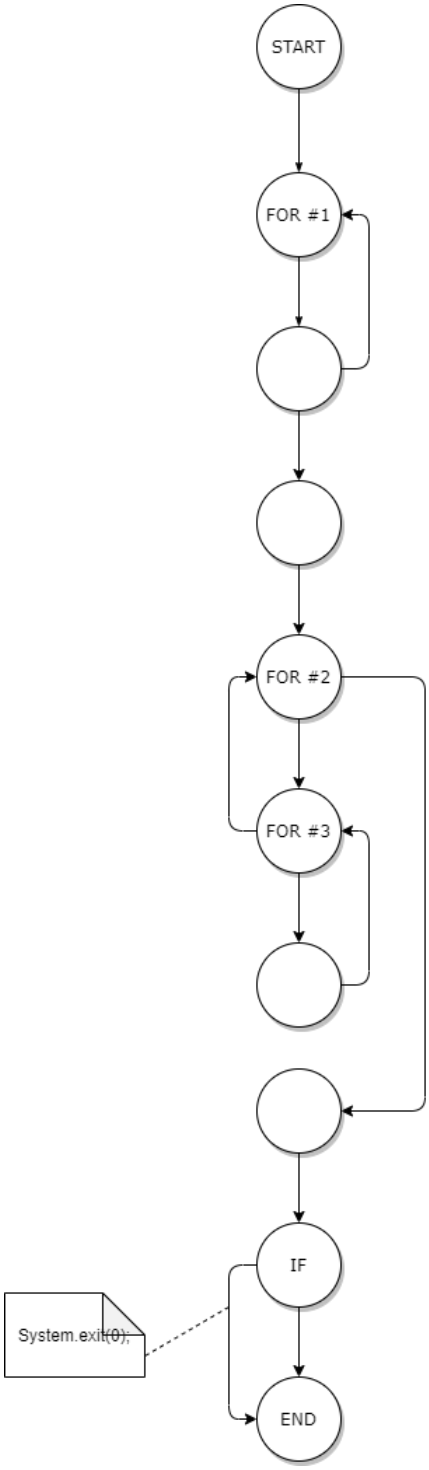
Longitud (N) = $N_1 + N_2 = 8 + 12 = 20$
Vocabulario (n) = $n_1 + n_2 = 34 + 56 = 90$
Volumen (V) = $N \cdot \log_2(n) = 20 \cdot \log_2(90) = 129{,}83$

4.3.3 Función main().

4.3.3.1 Número de líneas de código.

En esta función se contabilizan 23 líneas de código.

4.3.3.2 Complejidad ciclomática.



4.3.3.3 Halstead. Tokens de la función.

Para esta función, se contabilizan 240 tokens entre los operadores y operandos. Los tokens son los siguientes:

Operandos	Nº	Operadores	Nº
main	1	()	42
args	1	String	1
scan	6	[]	20
“Flood Fill Test\n”	1	{}	4
“Enter dimensions of grid”	1	Scanner	2
M	4	=	11
N	3	new	3
arr	5	System	7
i	8	.	20
0	3	in	1
2	3	;	22
‘O’	1	out	5
“Enter grid with (…) for obstacle”	1	println	5
1	4	int	7
j	4	nextInt	4
“Enter coordinates(…)”	1	char	2
sr	3	+	5
sc	3	for	3
‘P’	1	++	3
“Invalid coordinates”	1	Arrays	1
FloodFill	2	fill	1
ff	2	<	3
fillGrid	1	next	1
		charAt	1
		exit	1
		if	1
		!=	1
		,	3
TOTAL	**60**	**TOTAL**	**180**

Tabla 4. Tokens de la función main().

Longitud (N) = $N_1 + N_2 = 8 + 12 = 20$
Vocabulario (n) = $n_1 + n_2 = 34 + 56 = 90$
Volumen (V) = $N \cdot \log_2(n) = 20 \cdot \log_2(90) = 129{,}83$

4.3.4 Total del programa.

4.3.4.1 Número de líneas de código.

Existen 45 líneas de código totales.

4.3.4.2 Halstead. Tokens de la función.

El programa cuenta con 442 tokens totales.

Operandos	Nº	Operadores	Nº
Func. Main	60	Func. Main	180
Func. fillGrid	34	Func. fillGrid	56
Func. Display	20	Func. Display	66
FloodFill	1	Import	2
		Java	2
		.	4
		util	2
		Scanner	1
		;	2
		Arrays	1
		Public	2
		Class	1
		{}	2
		Private	2
		Void	3
		static	1
TOTAL	**115**	**TOTAL**	**327**

Tabla 5. Tokens totales del script en Lizard.

4.4 Conclusiones.

Lizard ayuda mucho a conseguir resultados rápidamente sobre complejidad de cualquier script. Puede acompañar a cálculos hechos a mano para despejar dudas o ahorrarse tiempo.

Se considera una herramienta muy útil para cualquier proyecto, no tiene limitaciones de longitud del lenguaje y también hay que destacar la cantidad de lenguajes de programación que permite calcular. Es una herramienta esencial para cualquier estudio rápido sobre la complejidad de un proyecto de software.

Capítulo 5. SonarQube: Análisis de código

5.1 ¿Qué es SonarQube?

Sonarqube es una herramienta de revisión automática de bugs, vulnerabilidades y problemas más profundos en el código. Se puede integrar a un flujo de trabajo ya existente como GitHub o AzureDevs para la continua inspección del código que se vaya creando a través de un IDE.

Su funcionamiento es el siguiente [14]:

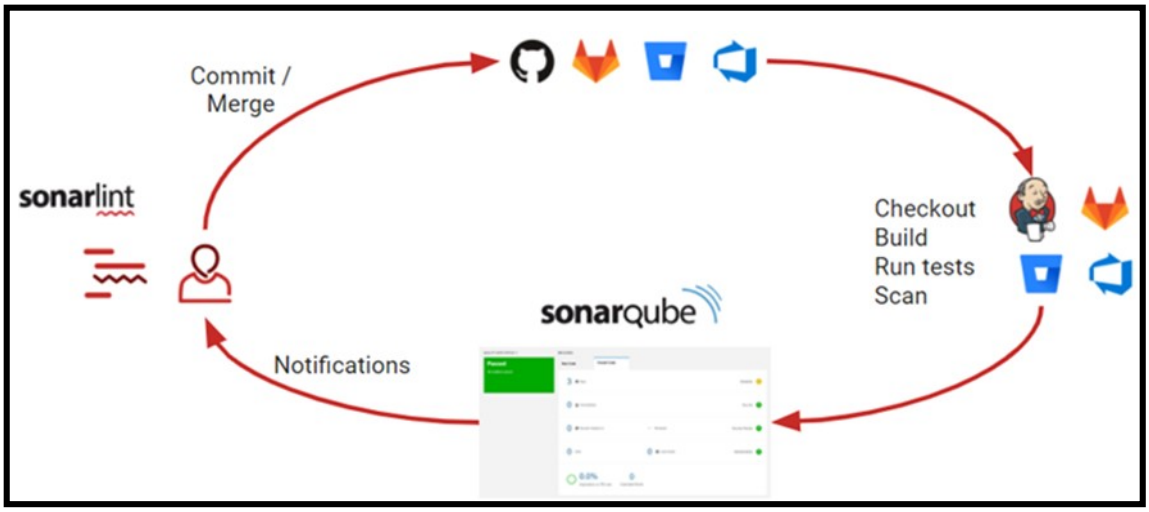


Figura 31. Funcionamiento de SonarQube esquemáticamente.

1. Los programadores desarrollan el código en su IDE (se recomienda el uso de SonarLint para recibir los comentarios oportunos directamente en el editor).
2. Las herramientas de integración continua como pueden ser GitHub, GitLab, Bitbucket o Azure DevOps comprueban, crean y ejecutarán pruebas unitarias y SonarQube comprobará los resultados.
3. Estos resultados se publican en el servidor de SonarQube para poder comunicárselos a los desarrolladores a través de su interfaz.

En el caso de este documento, se utilizarán la herramienta GitHub con la integración de SonarCloud.

5.1.1 SonarCloud.

SonarQube contiene una integración en la nube llamada SonarQube que puede ser utilizada en GitHub, Bitbucket, Azure DevOps y GitLab.

Su servicio en la nube tiene coste para las empresas y es gratis para los proyectos públicos:

Condiciones para los proyectos públicos:

- Uso de líneas de código ilimitado.
- Cualquier persona puede ver y explorar el proyecto.
- Acceso completo a las herramientas de SonarCloud.
- Se pueden crear equipos de trabajo con las personas que se deseen.

El precio para las empresas depende del número de líneas de código que contenga el proyecto:

Up to lines of code	Price per month in €
100k	10
250k	75
500k	150
1M	250
2M	500
5M	1500
10M	3600
20M	5000
30M	6250

Figura 32. Tabla de precios de SonarCloud.

Para más información sobre los precios para este tipo de casos, se encuentra en el siguiente link: https://sonarcloud.io/pricing.

5.2 Instalación de SonarCloud en GitHub.

La primera ver que se visita SonarCloud, ofrece a sus usuarios las integraciones que éste ofrece. En este caso, seleccionar “GitHub”:

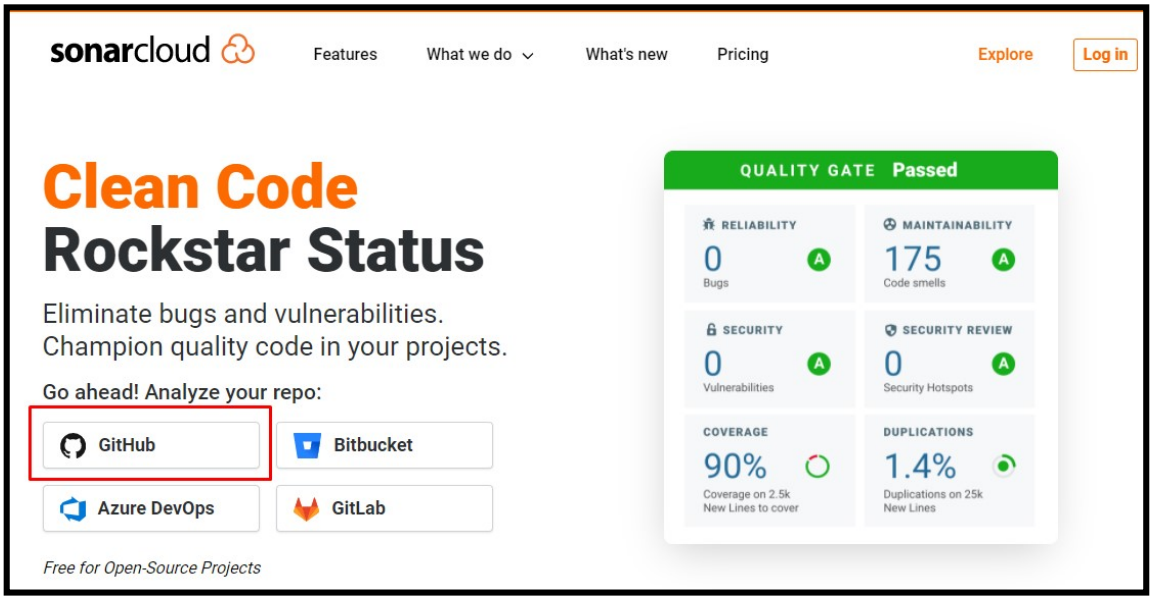


Figura 33 . Página principal de SonarCloud.

5.2.1 Integración de GitHub en SonarCloud.

Iniciar sesión con la cuenta de usuario dónde se encuentren los repositorios a analizar, dando permiso a SonarCloud para poder acceder a ellos más tarde:

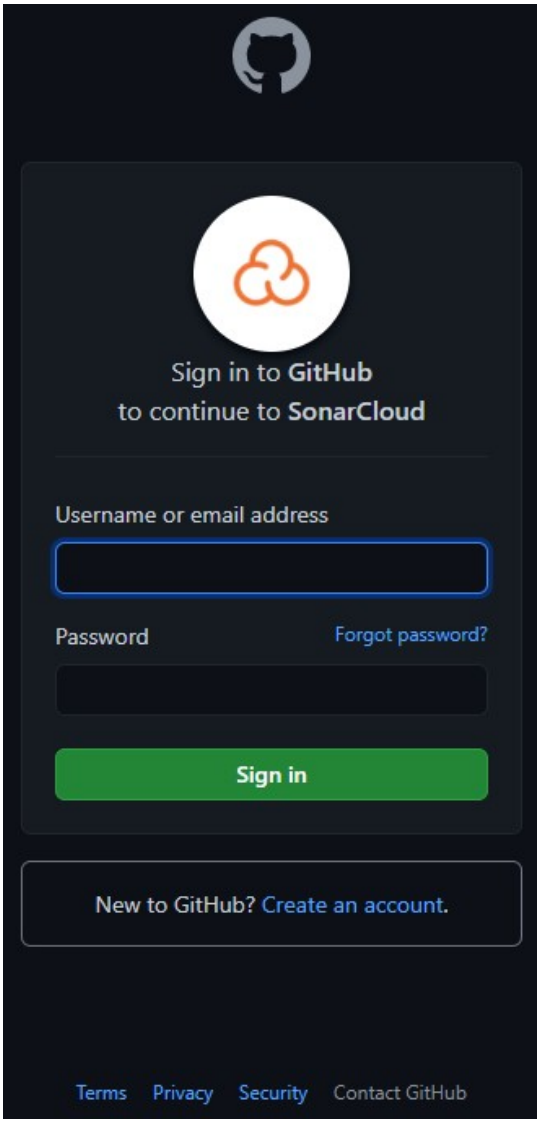


Figura 34 . Inicio de sesión en GitHub desde SonarCloud.

Seleccionar los repositorios que serán analizados, debe seleccionarse al menos uno:

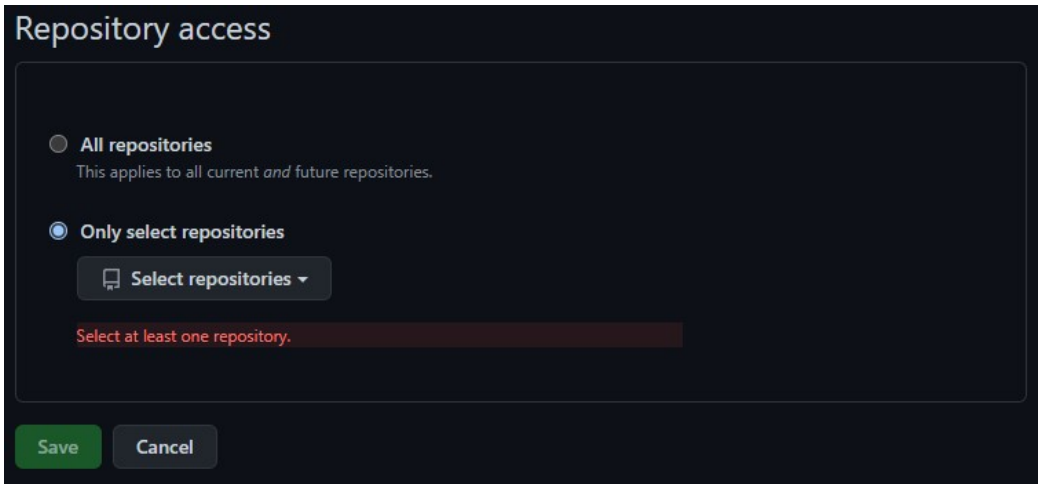


Figura 35 . Selección de repositores en SonarCloud.

Se creará una organización dónde se encontrarán más tarde los proyectos que se analizarán por SonarCloud.

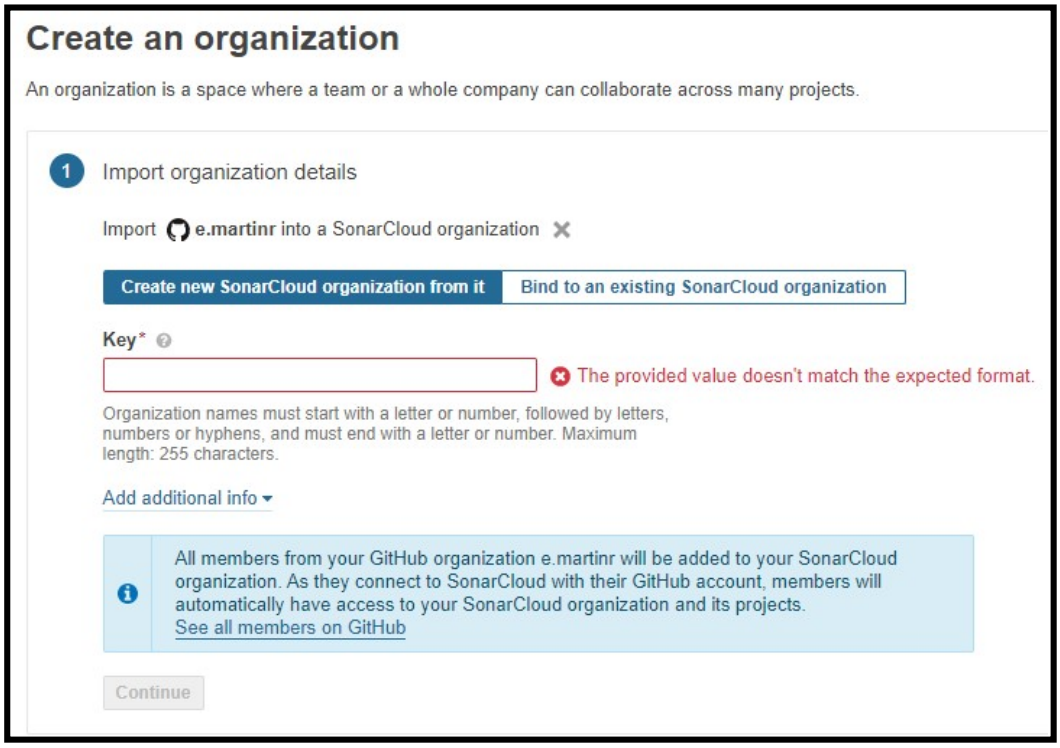


Figura 36. Creación de la organización en SonarCloud con GitHub.

Si es el primer proyecto que se analiza en SonarCloud, seleccionar "Analyze new Project":

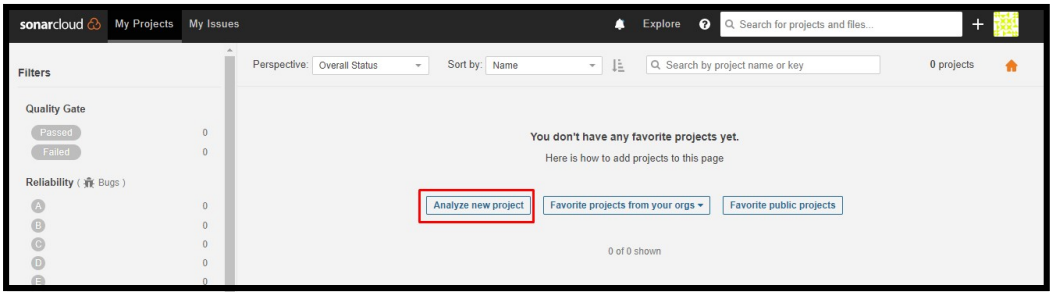


Figura 37. Página principal de SonarCloud con GitHub. Selección de proyectos.

A continuación, a parecerán los repositorios que se analizarán. Hay que tener en cuenta si el repositorio está desarrollado en estos lenguajes:

- ABAP
- Apex
- CSS
- Flex
- Go
- HTML
- JS
- Kotlin
- PHP
- Python
- Ruby
- Scala
- Swift
- TypeScript
- XML

Esto conllevaría a un análisis automático y SonarCloud lo analizaría después de haber seleccionado el repositorio sin ninguna configuración más que realizar. En el caso de contener un lenguaje que no aparezca en la lista, el análisis debería ser manual (apartado 6.2.2). Para más información sobre el análisis automático: https://sonarcloud.io/documentation/analysis/automatic-analysis/.

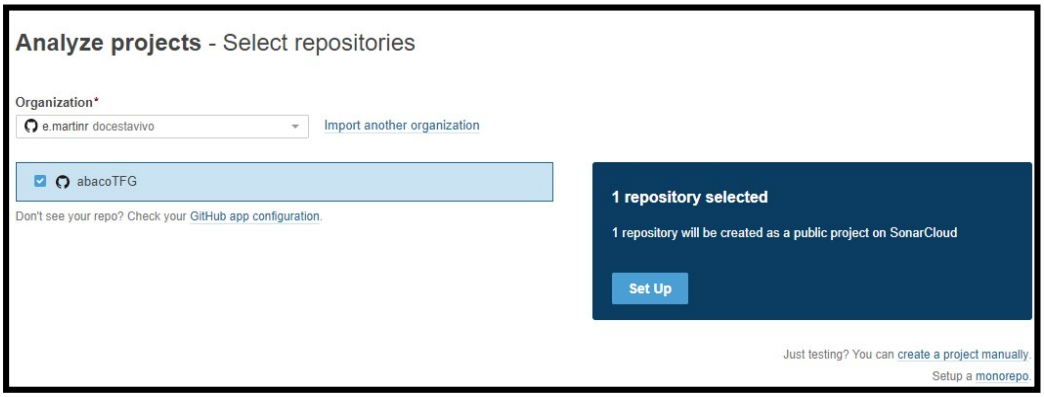


Figura 38. Selección de repositorios a analizar.

En el caso de existir un lenguaje no soportado por el análisis automático, SonarCloud mostraría esta información, ya que este repositorio está desarrollado en Java:

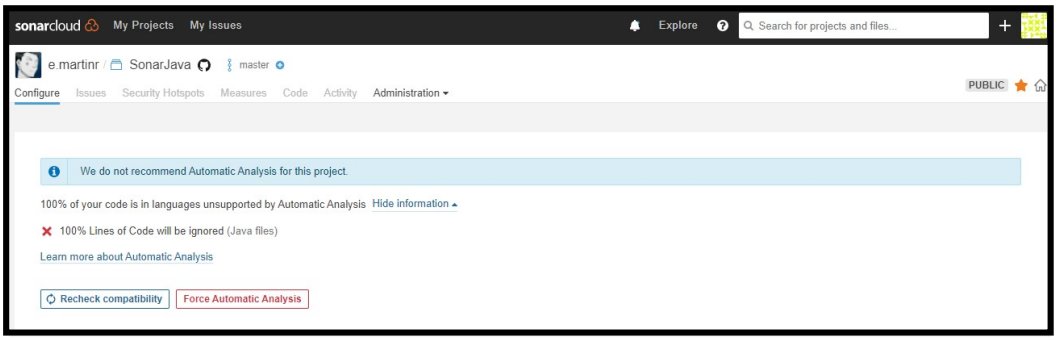


Figura 39 . Mensaje de SonarCloud que no puede realizar el análisis automático.

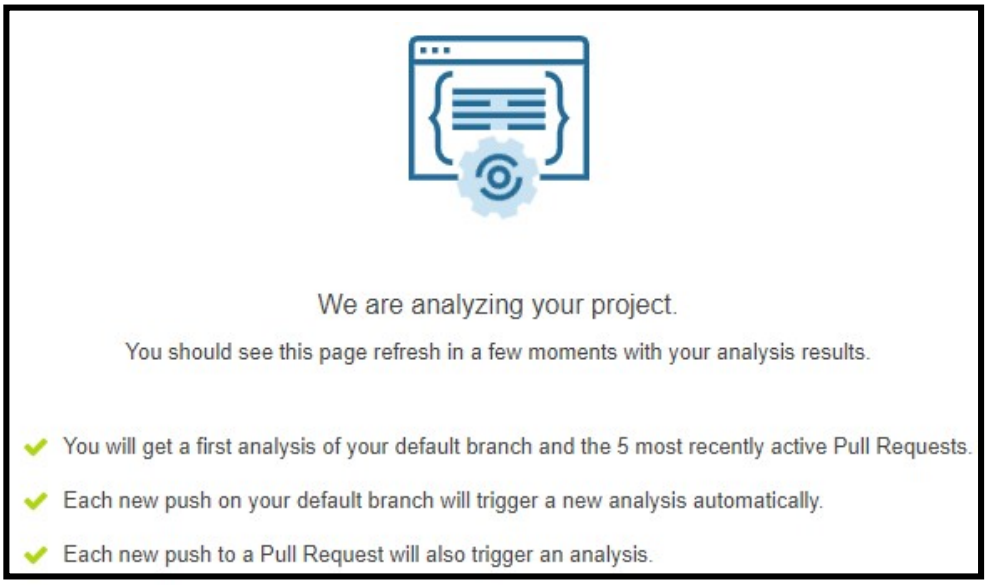


Figura 40. mensaje de SonarCloud en el caso de un análisis automático en Python.

5.2.2 Análisis manual para un repositorio en Java.

En la parte superior de la pantalla, seleccionar “Administration” y realizar estos pasos:

1. Seleccionar la opción “General Settings” y en la parte izquierda seleccionar “Languages” y después seleccionar el lenguaje en el que está desarrollado el proyecto:

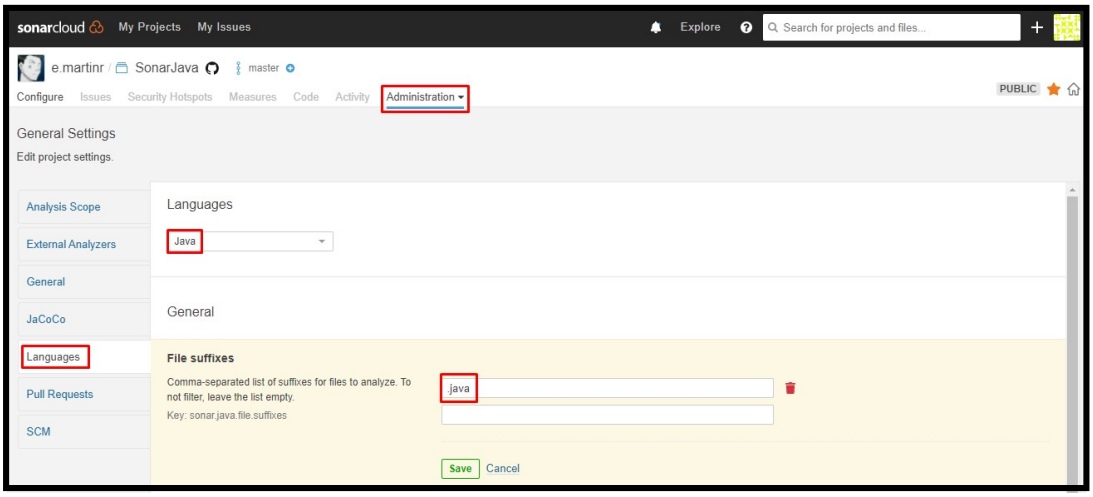


Figura 41. Configuración general de lenguajes del proyecto en SonarCloud.

En el apartado "File suffixes" deben estar escritas las extensiones de los archivos presentes en el proyecto, en el caso de este proyecto solo habrá ".java". Pulsar "Save".

2. Volviendo a "Administration", seleccionar "Branches & Pull Request". Aquí se debe señalar la rama que se analizará del proyecto.

3. De nuevo en "Administration", seleccionar "New Code" y elegir la opción que más convenga realizar a la hora de haber nuevo código en el repositorio, para que el análisis sea también nuevo.

4. Por último, de nuevo en "Administration" seleccionar "Update Key". Si la key no se llama igual a como se llama el repositorio en GitHub, debe modificarse, ya que este error es muy común y puede generar problemas futuros.

Figura 42. Modificación de la Key del repositorio en SonarCloud.

Queda aclarar que, si se desea borrar el repositorio de los proyectos de SonarCloud, en "Administration" seleccionar la última opción "Deletion" que borrará el proyecto. No se borrará el repositorio de GitHub, solo de SonarCloud.

Cuando ya se han realizado estos cambios, seleccionar "Configure" en la barra superior y en la parte inferior aparecerán varias opciones para elegir un método de análisis. En el caso de GitHub, se recomendará "With GitHub Actions". Cuando se selecciona, aparecerán los pasos a seguir para implementar este método.

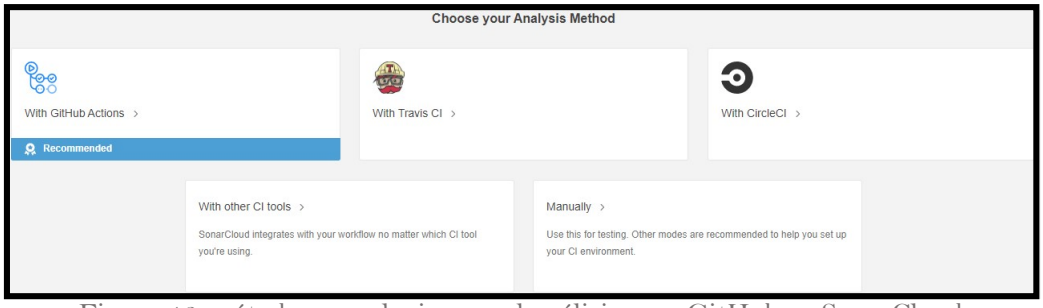


Figura 43. métodos que elegir para el análisis para GitHub en SonarCloud.

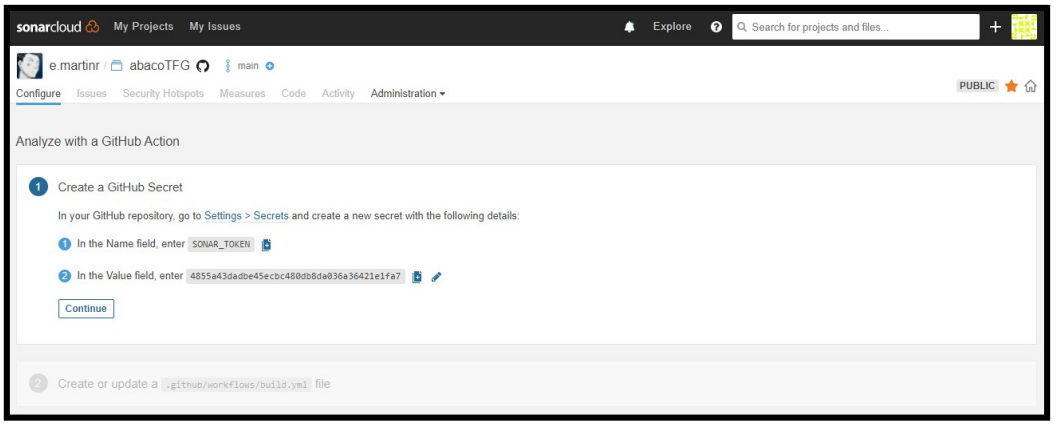


Figura 44. Pasos que realizar para utilizar Git Actions como método de análisis en SonarCloud.

1. Crear un secreto en GitHub. Desde el repositorio, seleccionar “Settings”:

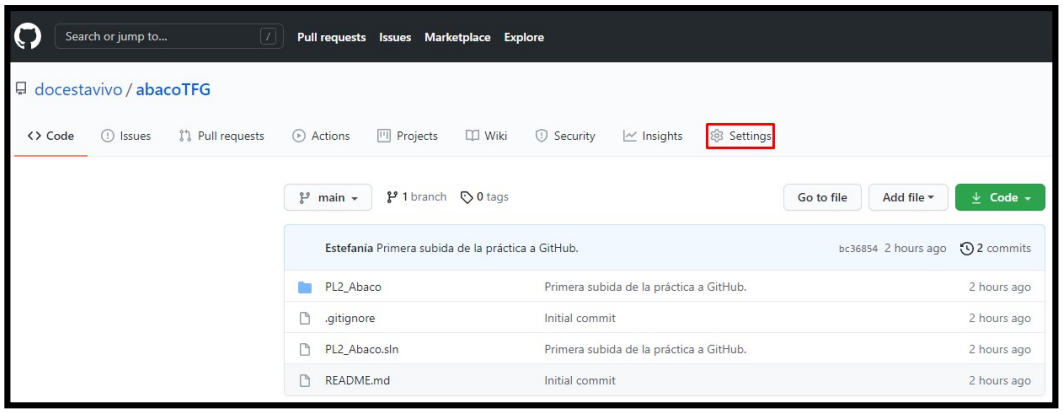


Figura 45. Configuración de un repositorio en GitHub.

En las opciones de la izquierda, seleccionar “Secrets” y después “New repository secret”.

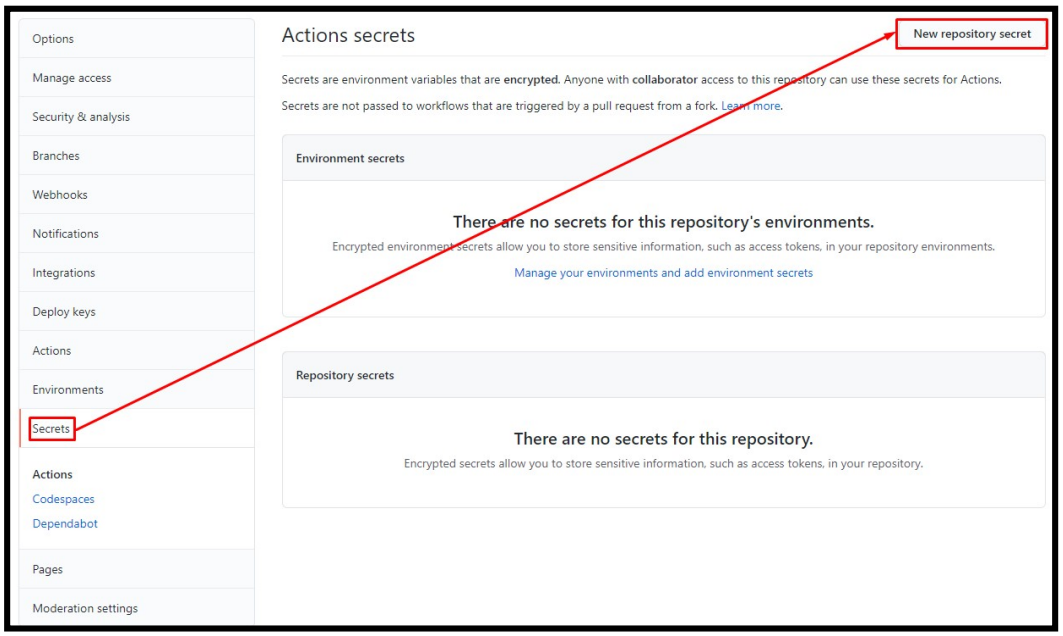


Figura 46. Botón de creación de un secreto en un repositorio en GitHub.

El nombre y el valor del secreto serán los que indique SonarCloud.

2. Saltar el paso 2 en SonalCloud y pasar directamente al 3, ya que antes de realizar los cambios que se describen en el paso 2, que es el más importante, se debe realizar este paso primero:

   a. Crear un archivo nuevo llamado “sonar-project.properties” y pegar lo que indique SonarCloud más la siguiente línea de código después de las dos primeras:

   ```
   sonar.java.binaries=RUTA_DEL_CODIGO_JAVA
   ```

   Por ejemplo, en este repositorio, el código se encuentra en la carpeta “src”, con que la línea sería “`sonar.java.binaries=src`”. En el caso de haber varias, separarlas con una coma.

   Prestar atención a que “sonar.projectKey” coincida con el nombre del repositorio en GitHub. En el caso de no ser así volver a los pasos anteriores en la pestaña Administration > Update Key.

3. Volviendo al paso 2 de SonarCloud, modificar el archivo .github/workflows/build.yml. Si no se han creado ya GitHub Actions, este archivo se encontrará oculto, y no se puede modificar como tal. Este archivo contiene las acciones a realizar cuando se ejecute el programa. Esto en GitHub se conoce como GitHub Actions. Es la misma idea que sigue, por ejemplo, Azure Pipelines.

Para crear el archivo seleccionar “Actions” en la barra superior del repositorio. Si ya se tiene creado un GitHub Action, añadir lo que indica SonarCloud, seleccionando en la parte superior el build más adecuado al código, en el caso de Java sería el último “Other (for JS, TS, Go, Python, PHP…)”.

En el caso de ser el primero, basta con crearlo uno mismo seleccionando esta opción en GitHub Actions y copiando todo lo que hay en SonarCloud borrando todo lo que hubiera antes:

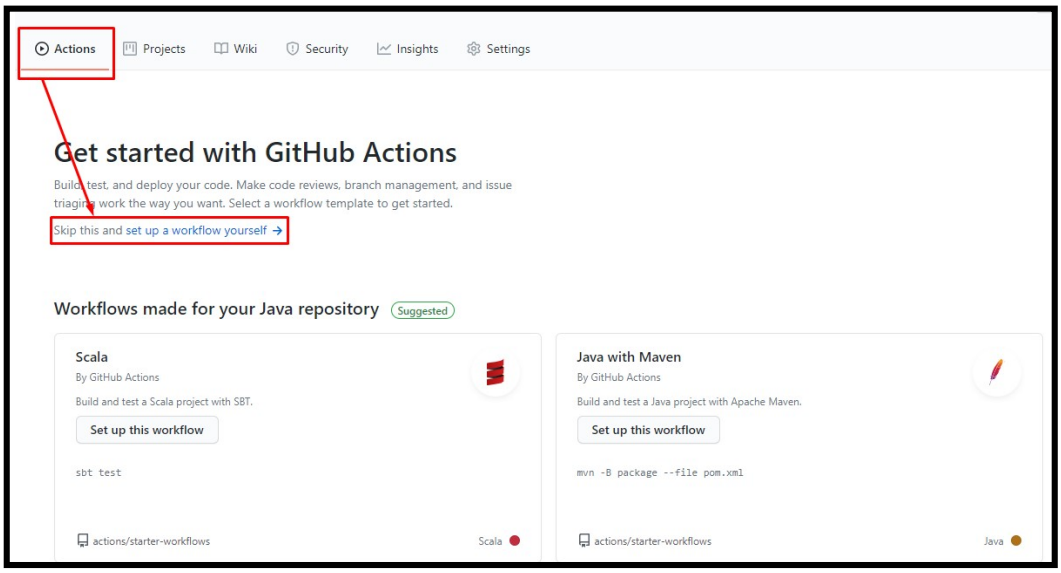


Figura 47. Creación de un GitHub Action.

No olvidar cambiar el nombre del archivo por "build.yml" en la parte superior del editor del GitHub Action:

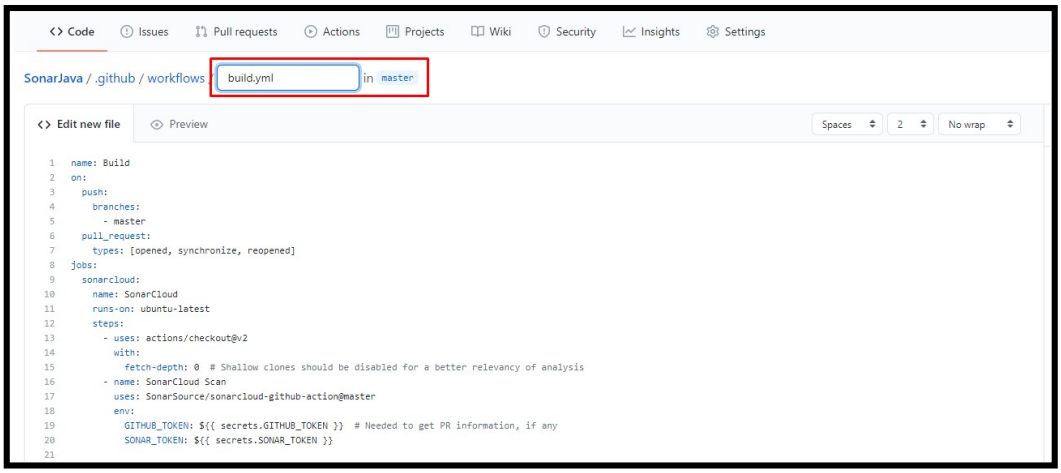


Figura 48. Cambio de nombre del GitHub Action en su editor.

Cuando ya se haya realizado todo, pulsar “Start commit” en la parte derecha y realizar el commit del nuevo archivo.

Volviendo a “Actions” en la barra superior, se podrá ver que ya se ha creado un workflow que realizará todo lo que se ha escrito en “build.yml”:

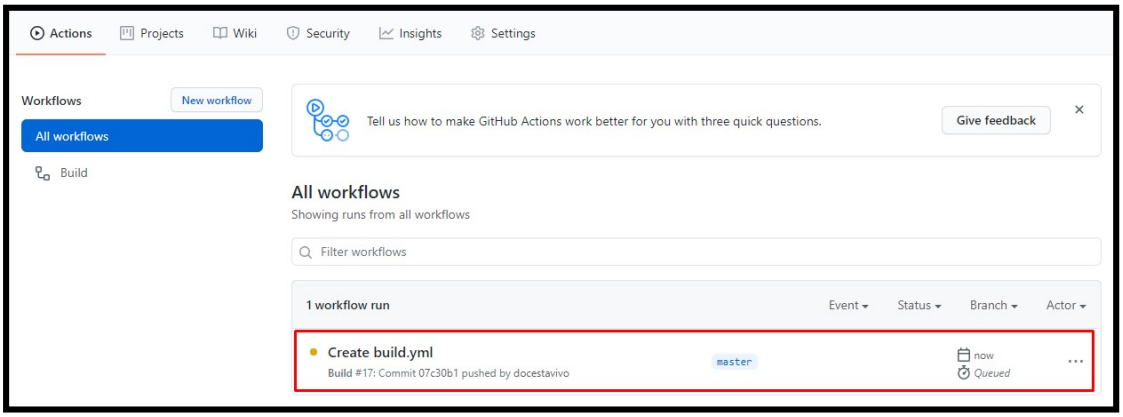


Figura 49. Workflow existentes en el repositorio GitHub.

Para ver el estado en el que se encuentra el workflow, seleccionar el título de este, que llevará a la siguiente pantalla:

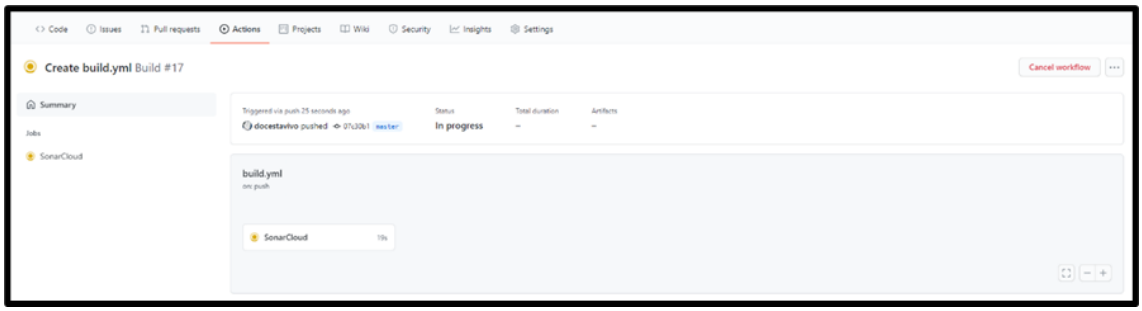


Figura 50. progreso actual del workflow.

Si se selecciona “SonarCloud” se podrá ver en detalle el proceso que esté llevando a cabo:

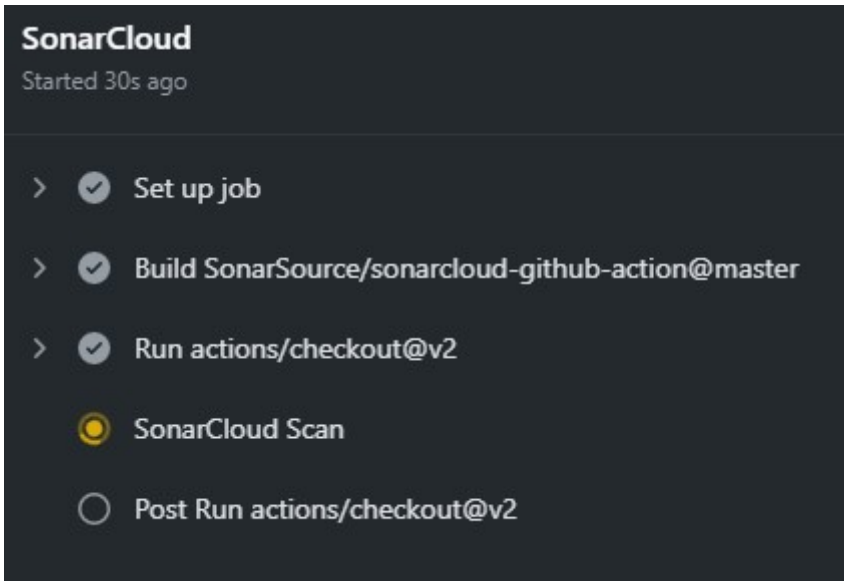


Figura 51. Procesos actuales del workflow en GitHub Actions.

Esto tomará uno o dos minutos, hasta que finalmente se muestre de esta manera si se han seguido todos los pasos para este repositorio desarrollado en Java:

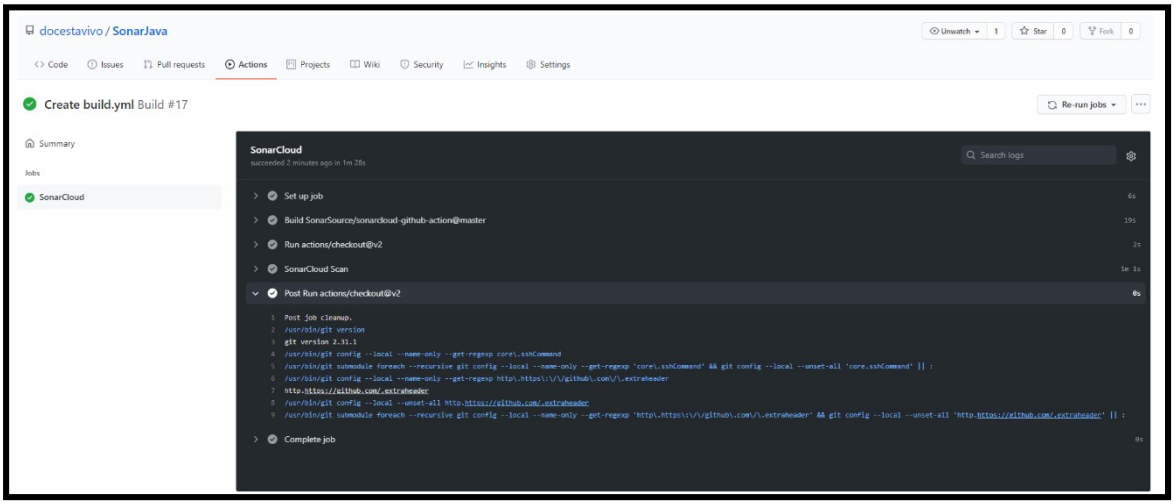


Figura 52. Resultado correcto del workflow en Java.

En el caso de encontrarse algún problema, los círculos se encontrarán de color rojo y si se expanden muestran los logs de todas las tareas que se han realizado. Es posible que algún problema que se encuentre coincida con uno de los errores que se describen en este apartado (HIPERVINCULO).

Si se vuelve a SonarCloud, se mostrarán los resultados del análisis del repositorio:

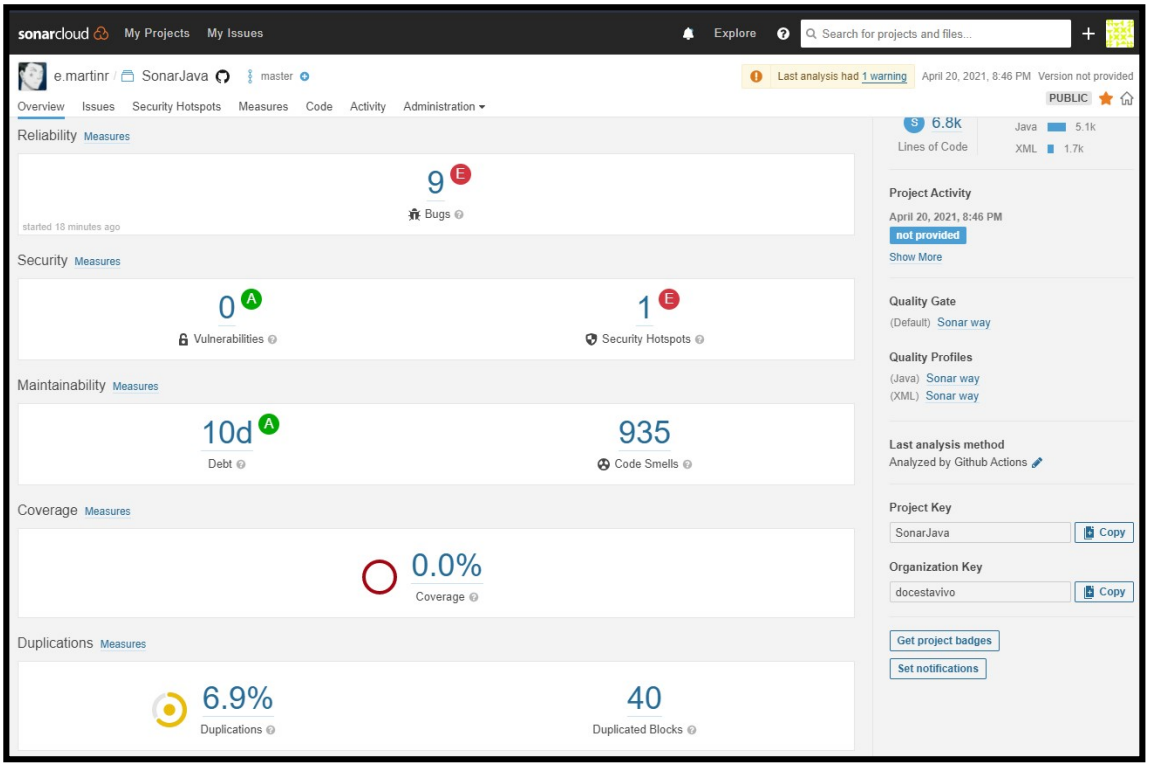


Figura 53. Resultados del análisis del proyecto en Java.

En el caso de no mostrarse cuando ha ido todo correctamente, actualizar la página.

5.3 Interpretación del análisis en SonarCloud.

Las medidas presentadas por SonarCloud al pulsar en "measures" en cada uno de los resultados son las siguientes [15]:

5.3.1 Reliability (Fiabilidad).

- Número de bugs. Un bug [16] es un error en el código que rompe la ejecución del código y debe ser solucionado inmediatamente.

- Nivel de fiabilidad:
  - A = cero bugs
  - B = al menos un bug poco importante
  - C = al menos un bug importante
  - D = al menos un bug crítico
  - E = al menos un bug de bloqueo (aquellos que si se encuentran presentes es imposible realizar un despliegue del proyecto [17]).

- Reliability remendation effort: El tiempo que tomará solucionar todos los bugs encontrados. Si el valor es en días, los días se consideran de 8 horas.

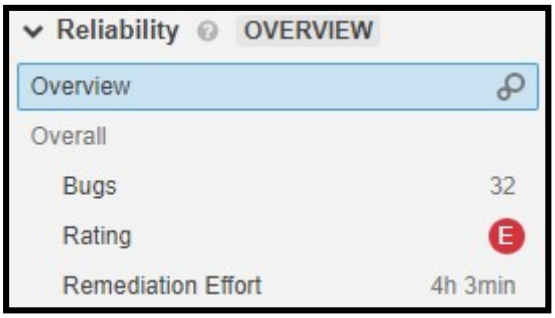


Figura 54. Ejemplo de medidas de Reliability del rpoyecto Hero Of Antair (pruebaSonarPython2).

¿Cómo considera Sonarqube un bug?

Las reglas que sigue SonarQube para considerar en el código un posible bug, se encuentra en la siguiente dirección: https://rules.sonarsource.com/. En esta página web se separan los posibles problemas analizados por lenguaje de programación, separándolos por el tipo de análisis: vulnerabilidades, bugs, security hostposts y code smells. Y cada uno de ellos los separa por categorías: critical, major, minor. E incluso etiquetas (tags). También da la opción de crear reglas propias.

Ejemplos:
Cuando aparece un bug por parte de Sonarqube, lo muestra en la línea de código afectada. Si se selecciona, se mostrará con un título descriptivo del problema y debajo el tipo de error (bug, code smell), el rating del error, el estado actual y el esfuerzo en resolverlo.

A la derecha del título del error, hay un link “Why is this a issue?” que muestra en más detalle el error en cuestión.

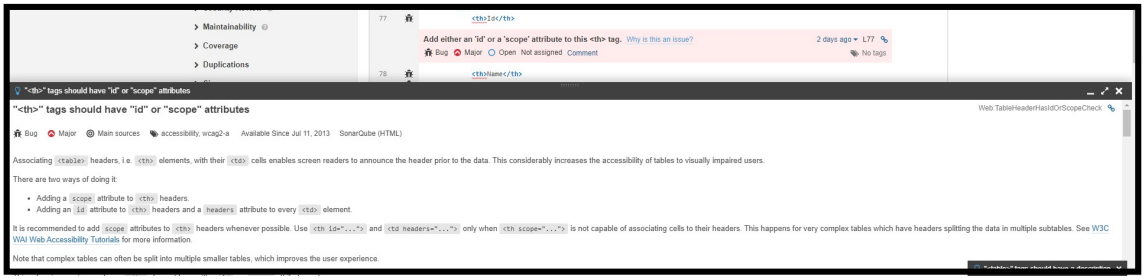


Figura 55. Información sobre un issue al seleccionar "Why is this a issue?".

Estos detalles corresponden con la documentación anteriormente citada.

A continuación, se muestran diferentes bugs posibles:

Figura 56. Bug de categoría "minor" encontrado de la aplicación pública Visualpahit Vprofile Webapp.

Figura 57. Bug de categoría "major" encontrado en la aplicación pública mediawiki-core.

Figura 58. Bug asignado a un usuario en la aplicación pública policy-gui.

5.3.2 Security (Seguridad).

- Número de vulnerabilidades.

- Nivel de seguridad:
  - A = cero vulnerabilidades.
  - B = al menos una vulnerabilidad poco importante.
  - C = al menos una vulnerabilidad importante.
  - D = al menos una vulnerabilidad critica.
  - C = al menos una vulnerabilidad de bloqueo.

- Security remediation effort: El tiempo que se tomará para solucionar las vulnerabilidades. Si el valor es en días, los días se consideran de 8 horas.

- Número de Security Hotspots: los security Hoyspots son [18] un fragmento en el código sensible a la seguridad que el desarrollador debe revisar. La diferencia con una vulnerabilidad es que las vulnerabilidades deben resolverse enseguida ya que afectan a todo el proyecto. Un hotspot es un fragmento en el código que debe revisarse y no tiene por qué afectar a todo el proyecto y dependerá del desarrollador darle una solución o no.

- Nivel de security review: en el porcentaje de los Security Hotspots revisados.
  - A: mayor o igual al 80%.
  - B: entre 70% y 80%.
  - C: entre 50% y 70%.
  - D: entre 30% y 50%.
  - E: menos del 30%.

- Número de Security Hotspots revisados.

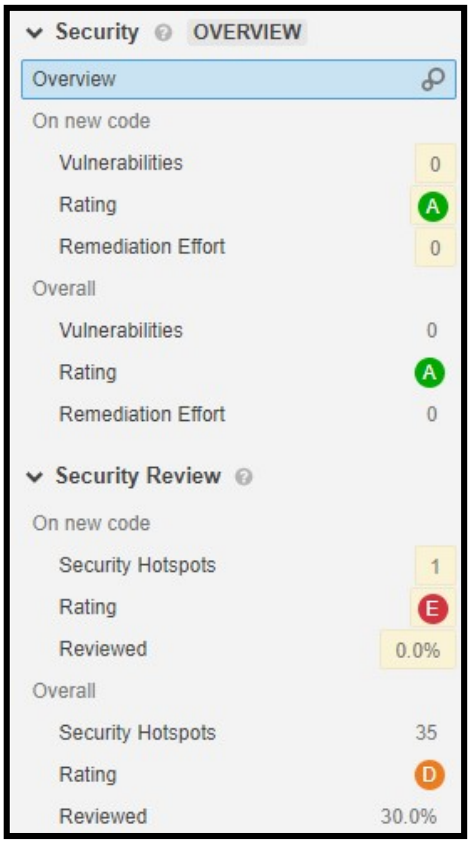


Figura 59. Ejemplo de medidas de seguridad del navegador Brave (brave-core).

Las vulnerabilidades se detectan de la misma manera que los bugs. A continuación, se citan varios ejemplos encontrados en aplicaciones públicas:

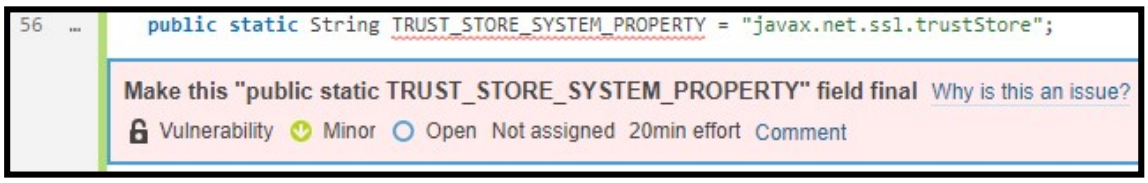


Figura 60. Ejemplo de una vulnerabilidad de categoría "Minor" en la aplicación pública Fabric8::Kubermetes Client.

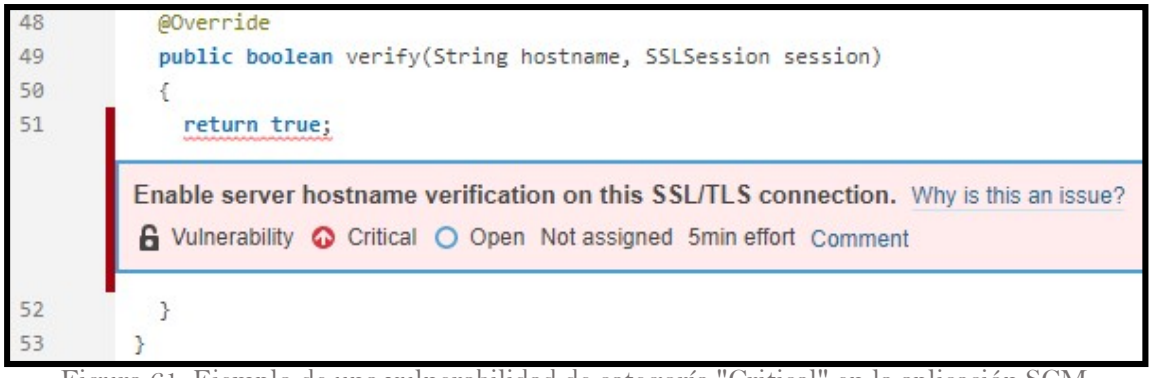


Figura 61. Ejemplo de una vulnerabilidad de categoría "Critical" en la aplicación SCM-Manager.

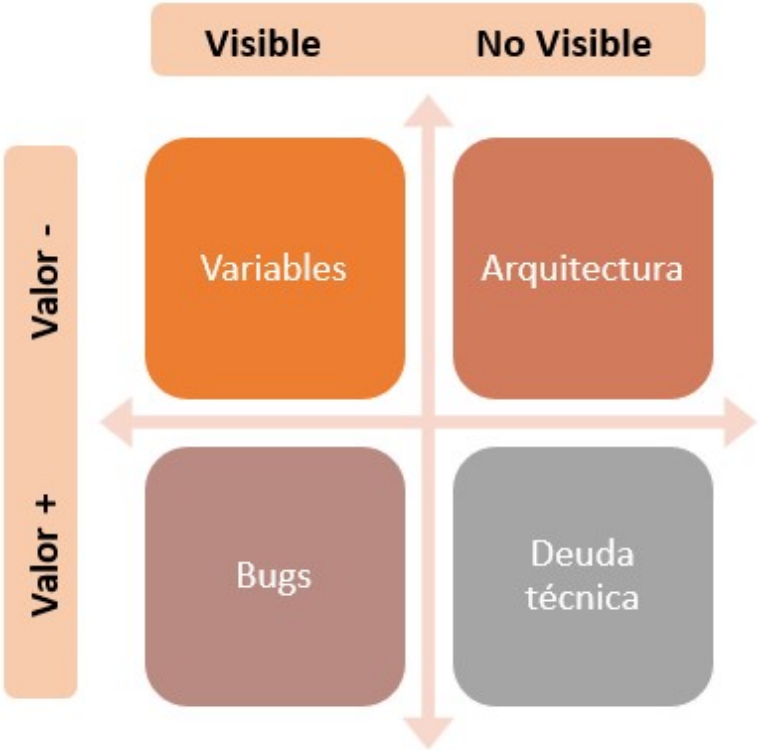
Figura 62. Ejemplo de una vulnerabilidad de categoría "Blocker" en la aplicación pública SCM-Manager.

5.3.3 Maintainabilty (Mantenibilidad).

5.3.3.1 Deuda técnica.

La deuda técnica es [19] la consecuencia de haber un desarrollo pobre en el software.

Puede representarse como algo invisible y de valor negativo en el proyecto, de manera que, si no se tiene control sobre ella, el control de seguimiento sobre el código puede terminar resultando insostenible a largo plazo. Sin embargo, si se tiene un control sobre ella, puede darse el caso de resultar un beneficio a corto plazo para el proyecto negociando su calidad de software a largo plazo.



¿Qué causa la deuda técnica? [19]

- Malas prácticas de diseño y codificación.
- Mala organización de las funcionalidades del sistema y estimación del proyecto.

También se tienen en cuenta estos tipos de deudas que pueden causarla:

- **Deuda estratégica:** se trata de una deuda a largo plazo debida a decisiones estratégicas de la empresa de forma proactiva.

- **Deuda táctica:** esta deuda a coroto plazo de forma reactiva se da lugar, por ejemplo, en decisiones que tendrá un resultado optimo más adelante. Un caso concreto es cuando se quiere implementar una función que aún no está del todo desarrollada y se despliega incompleta con la intención de completarla más adelante.

- **Deuda incremental:** es la deuda más común de todas. Se debe a las causas antes mencionadas, ausencia de comentarios o comentarios dispersos, ausencia de homogeneización… Si no se controlan estos aspectos, puede darse el caso de acumularse.

Los tipos de deuda técnica corresponden con cada etapa de la calidad de software [19]:

- **Deuda de requisitos:** se trata de la especificación e implementación óptima de los requisitos existentes en el proyecto. Esto implica que están claramente definidos ya que de ellos depende el éxito del proyecto.

- **Deuda de código:** se encuentra en los code smells o en las malas prácticas. Esto se puede resolver con refactorización. Este tipo de deuda necesita atención ya que en la etapa de mantenimiento del proyecto puede ocasionar costes elevados, ya que implicaría esfuerzo mayor para implementar los cambios.

- **Deuda de diseño y arquitectura:** ocurre por decisiones tomadas de la arquitectura que afectan a los aspectos de calidad. Este tipo de deuda es importante ya que puede afectar a largo plazo al proyecto si no se llega a controlar.

- **Deuda de medio ambiente:** cuando el proyecto no se encuentra en un entorno adecuado, ya sea software, hardware o de su infraestructura.

- **Deuda de documentación:** ocurre cuando la documentación del proyecto incluye información desactualizada, incompleta o cuando hay demasiada.

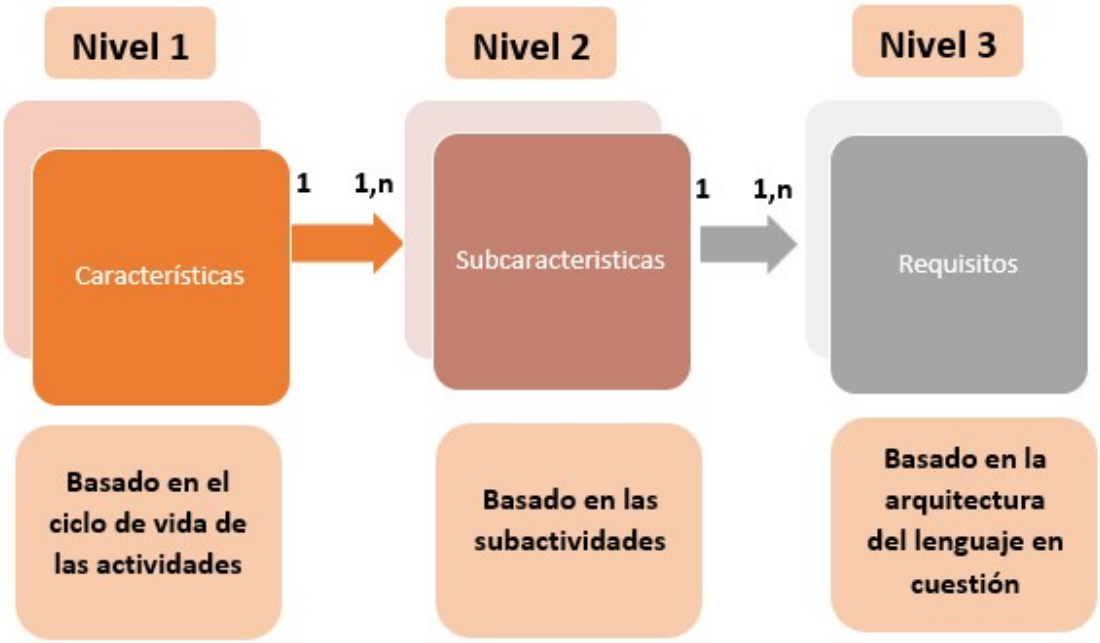
- **Deuda de pruebas:** es debido a la falta de scripts de prueba o a atajos en las pruebas.

SonarQube se centrará sobre todo en deudas relacionadas con las pruebas y el diseño del código.

5.3.3.2 SQUALE

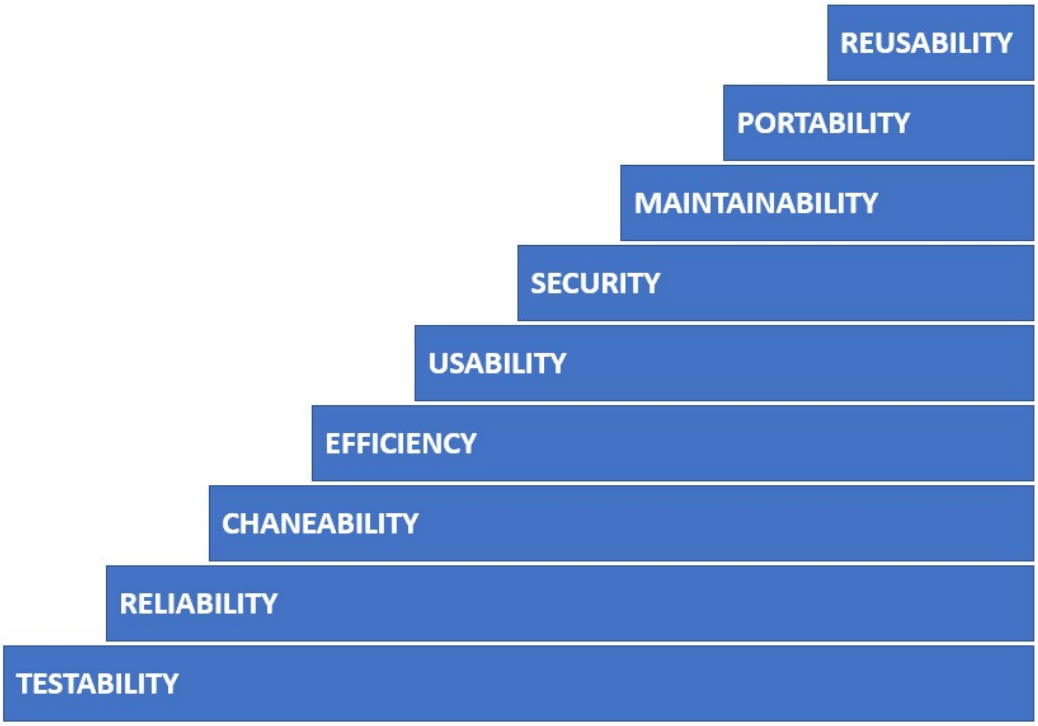
SonarQube utiliza un modelo de análisis llamado SQUALE (Software Quality Assessment base don Lifecycle Expectations) que organiza los requisitos no funcionales sobre mantenibilidad, testabilidad, fiabilidad, portabilidad, reusabilidad, eficiencia y seguridad. Para calcular la deuda técnica se basa en los estándar ISO 9126 y 25000. [19]

SQUALE está formado en tres estructuras: Características, subcaracterísticas y requisitos.



5.3.3.2.1 Nivel 1: Características.

Las características de SQUALE corresponden con las actividades que ocurren en el ciclo de vida del software que se describen en el estándar ISO 25010:



- **Testability:** se crean criterios de prueba para que éstas muestren si el producto cumple con esos criterios.
- **Reliability:** capacidad del código de cumplir las funciones especificadas cuando funcione bajo ciertas directrices durante un periodo de tiempo.
- **Changeability:** la capacidad que tiene el código de recibir cambios. Estos cambios pueden ser en el código, en el diseño o en la documentación.
- **Efficiency:** la capacidad de cumplimiento relativo a la cantidad de recursos utilizados bajo determinadas condiciones.
- **Usability:** la capacidad del proyecto de ser comprendido por el usuario final.
- **Capacidad de proteger** la información confidencial de usuarios o sistemas sin los privilegios necesarios para leerla o modificarla.
- **Maintainability:** la capacidad que tiene el código de poder modificarse efectiva y eficientemente por razones evolutivas, correctivas o perfectivas.
- **Portability:** capacidad del producto de ser transferido a un entorno software, hardware u operacional nuevo.
- **Reusability:** reciclaje del código, diseño o documentación en otros entorno, productos o sistemas.

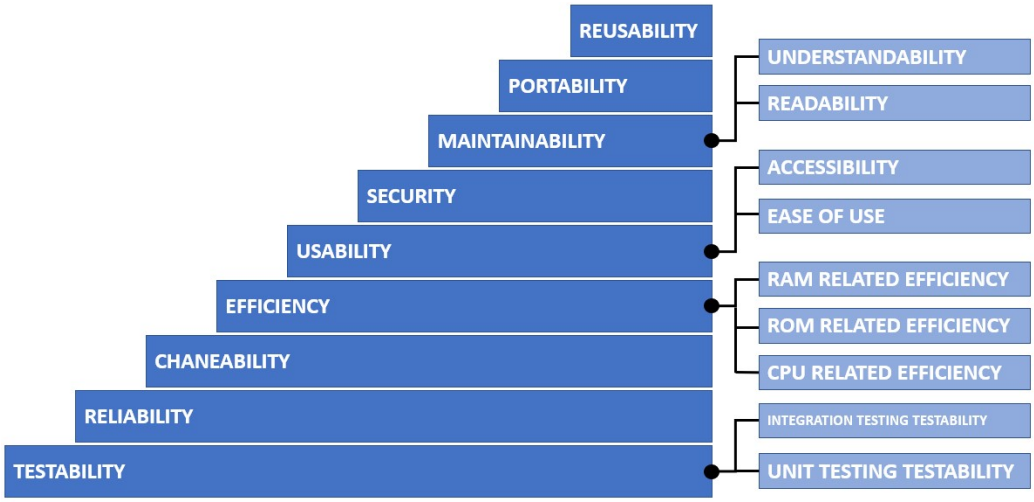
Ciertas características contienen varias subcaracteristicas.

5.3.3.2.2 Nivel 2: Subcaracteristicas.

Las subcaracteristicas consiguen que las características puedan realizar los análisis de manera más detallada y organizada. Cada subcaracteristica está relacionada con una sola característica. Hay dos tipos de subcaracteristicas:

- Las sucaracteristicas que corresponden con el ciclo de vida del código. Estas corresponden con las pruebas unitarias, las de integración, optimización del uso de recursos y optimización del tamaño del código.

- Las subcategorías que son resultado de las clasificaciones reconocidas como buenas y malas prácticas dependiendo del lenguaje, su arquitectura y codificación.



5.3.3.2.3 Nivel 3: Requisitos.

Los requisitos se refieren a los aspectos que existen en el código fuente, siempre respetando los criterios presentados en los principios fundamentales.

Cada requisito se enlaza con el nivel más bajo posible, relacionándose con la primera característica de calidad a la que contribuye cronológicamente.

A continuación de muestra un ejemplo de características, subcarasterísticas y requisitos existentes en Java.

Characteristic	SubCharacteristic	Generic Requirement Description
Maintainability	Understandability	No unstructured statements (goto, break outside a switch...)
Maintainability	Understandability	No use of "continue" statement within a loop
Maintainability	Understandability	File comment ratio (COMR) > 35%
Maintainability	Readability	Variable name start with a lower case letter
Maintainability	Readability	The closing brace '}' is on a standalone line
Maintainability	Readability	The code follow consistant indentation rules
Maintainability	Readability	File size (LOC) <1000
Maintainability	Readability	No commented-out code
Efficiency	RAM related efficiency	Class depth of inheritence (DIT) <8
Efficiency	RAM related efficiency	No unused variable, parameter or constant in code
Changeability	Architecture related changeability	Class weighted complexity (WMC) <100
Changeability	Architecture related changeability	Class specification does not contains public data
Changeability	Logic related changeability	If, else, for, while structures are bound by scope
Changeability	Data related changeability	No explicit constants directly used in the code (except 0,1, True and False)
Reliability	Fault Tolerance	Switch' statement have a 'default' condition
Reliability	Logic related reliability	No assignement ' =' within 'if' statement
Reliability	Logic related reliability	No assignement ' =' within 'while' statement
Reliability	Logic related reliability	Invariant iteration index
Reliability	Data related reliability	No use of unitialized variables
Testability	Integration level testability	No "Swiss Army Knife" class antipattern
Testability	Integration level testability	Coupling between objects (CBO) <7
Testability	Unit Testing testability	No duplicate part over 100 token
Testability	Unit Testing testability	Number of ind. test paths within a module (v(G)) <11
Testability	Unit Testing testability	Number of parameters in a module call (NOP) <6

Figura 63 . Ejemplo en java de los niveles de SQUALE.

5.3.3.3 SQUALE en SonarQube.

SQUALE en SonarQube se basa en reglas y evidencias comunes en todos los lenguajes que son:

- Bloques duplicados.
- Pruebas unitarias que han dado error o no se han podido realizar.
- Cobertura de las pruebas escasa.
- Comentarios insuficientes.

La deuda técnica se mide en días y está medida por expertos. Si se tiene el plugin comercial SQUALE (https://www.sonarsource.com/plans-and-pricing/enterprise/), se pueden ajustar estos tiempos a la regla que se desee.

5.3.3.4 Calculo de la deuda técnica.

SonarQube tiene un valor determinado por cada concepto encontrado que dé paso a la deuda técnica.

En el artículo [19] de la bibliografía se diseña un caso de estudio para justificar estos cálculos que se muestran en este tipo de herramientas que muestran resultados sobre la deuda técnica. A continuación, se describe dicho caso de manera resumida utilizando solo una aplicación de las 8 que se utilizan en el estudio.

Se elige la aplicación número 8 del caso de estudio que tiene las siguientes características:

- Está desarrollado en Java, JavaScript y HTML.
- Su estilo arquitectónico es REST.

  REST (Tranferencia de Estado Representacional) [20] se trata de una arquitectura sofware para sistemas hipermedia distribuidos en Internet. Con esto se consigue mejor escalabilidad gracias a su protocolo cliente/servidor sin necesidad de que ni el cliente ni el servidor este al tanto del estado de las comunicaciones entre los mensajes existentes en el sistema. También contiene una sintaxis universal para identificar los recursos existentes.

- Su patrón de diseño es FACADE.
  El diseño Fachada o Facade [21] se trata de un patrón de diseño estructural de manera que reduce la complejidad de la división en subsistemas, minizando comunicaciones y dependencias entre estos.

- Tiene 768 líneas de código.

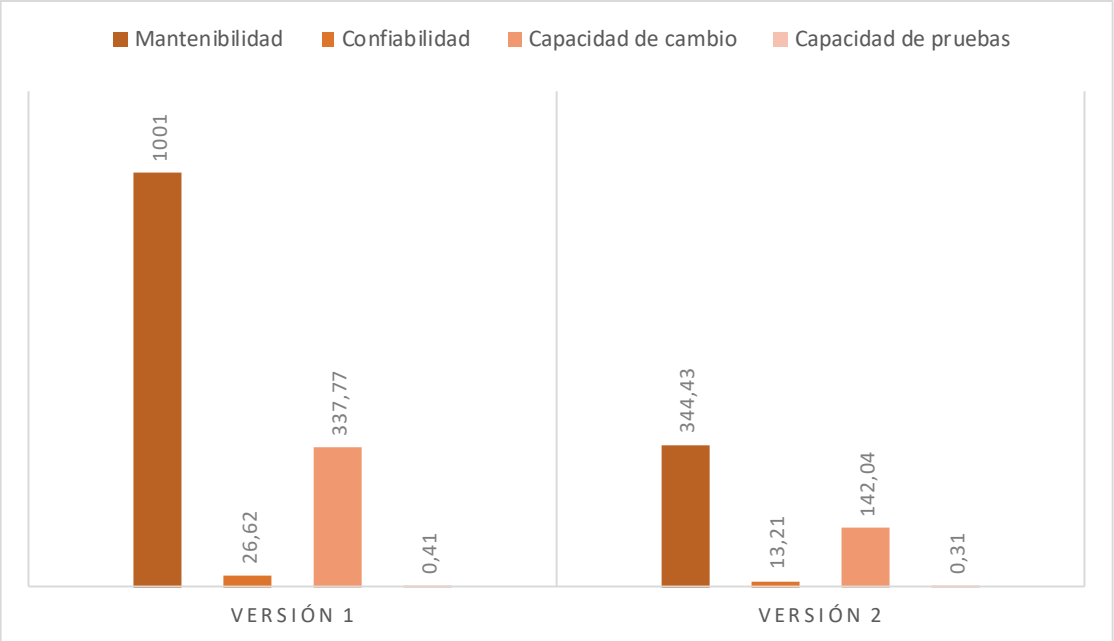
Se calculan las dos primeras actividades de la gestión de la deuda técnica: identificación y medición. En SonarQube se muestran los siguientes resultados:

- 252 violaciones en los 4 ficheros con 768 líneas de código.
- En la deuda técnica por características se muestran (8 horas al día):
  - Mantenibilidad: 2 días y 6 horas.
  - Confiablidad: un día.
  - Capacidad de cambio: 28 días.
  - Capacidad de pruebas: 30 minutos.
  - **Deuda total: 32 días.**

Se realizan cambios en la aplicación respondiendo a los cambios sugeridos por SonarQube y correcciones, creando una versión 1 (la original) y una nueva versión 2. Para esto se basa en criterios como:

- Severidad de la regla debe ser mayor o menor.
- El valor del costo de remediación total por regla debe ser a 50 minutos.
- Las violaciones no deben ser falsos positivos.

De esta manera, la deuda técnica total pasa a ser de 32 días a 2 días. Con esto se observa que la característica con mayor deficiencia es la mantenibilidad, ya que, al observar los resultados entre las ocho aplicaciones, se observa que la mantenibilidad baja un 65.5%, la confiabilidad baja un 50,37%, la capacidad de cambio baja un 57.94% y la capacidad de pruebas baja un 24.39%.



La característica con mayor costo es la capacidad de cambios, que conlleva a pensar que las ocho aplicaciones presentan problemas de evolución y en el futuro darán problemas a la hora de realizar cambios en su código.

La ecuación para el cálculo de cada característica es:

$$C = \sum_{SC=1}^{SC=n} Nº\ Violaciones\ x\ Costo\ Remediación$$

- C es la característica (Mantenibilidad, Confiabilidad, Capacidad de cambio o Capacidad de pruebas).
- SC es la subcaracterística que pertenece a la característica principal.

La Ecuación para la deuda técnica total sería la suma de todas las anteriores.

SQUALE implementa índices consolidados que se obtiene a partir de cada característica. De manera manual, los índices para las características anteriores serían:

- Consolidated Testability Index (SCTI) = SQALE Testability Index (STI)
- Consolidated Reliability Index (SCRI) = STI + SQALE Reliability Index (SRI)
- Consolidated Maintainability Index (SCMI) = STI + SQALE Maintainability Index (SMI) + SQALE Changeability Index (SCI)
- Consolidated Changeability Index (SCCI)= STI + SRI + SCI

5.3.3.5 Resultados en SonarQube.

- Número de code smells: Si los code smells [15] se dejaran como están, los cambios futuros en el código podrían llegar a ser más costosos ya que acarrearían más errores.

- Maintainability Rating: el valor de este rating dependerá del Technical Debt Ratio.

- Technical Debt [15] se trata del tiempo que se necesita para solucionar todos los asuntos de mantenibilidad o code smells.
  - A: menos del 5% del tiempo restante.
  - B: entre 6% a 10% del tiempo restante.
  - C: entre 11% a 20% del tiempo restante.
  - D: 21% a 50% del tiempo restante.
  - E: por encima del 50% del tiempo restante.

- Technical Debt: el esfuerzo necesario para solucionar los code smells. Si el valor es en días, los días se consideran de 8 horas.

- Techincal Debt Ratio: el resultado de este valor es obtenido a partir del coste del desarrollo del software y el coste de arreglarlo.

$$\frac{Remediation\ cost}{Development\ cost} = \frac{Remediation\ cost}{(0.06) * LOC}$$

- Remediation cost: el tiempo necesario para solucionar los asuntos de vulnerabilidad y fiabilidad.

- Development cost: el coste de una línea (0.06 días) por el total de todas las líneas de código.

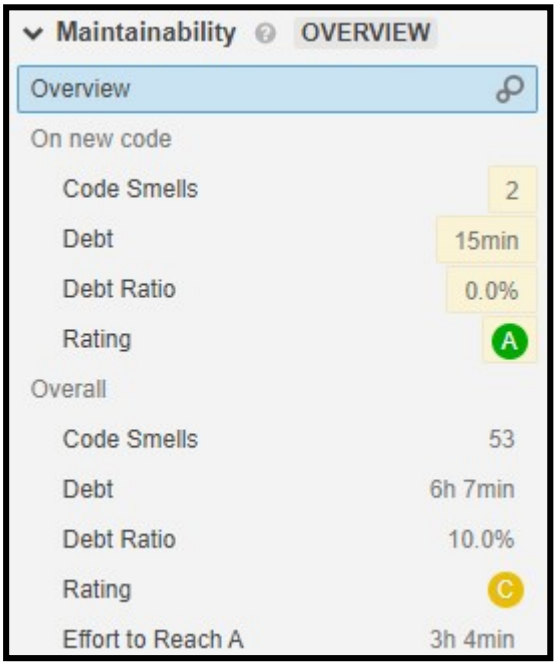


Figura 64. Mantenibilidad del proyecto Hearing Management Interface API.

Ejemplos encontrados en proyectos públicos con code smells:

Figura 65. Code smell de categoría "Major" en la aplicación pública externalapi-nbi.

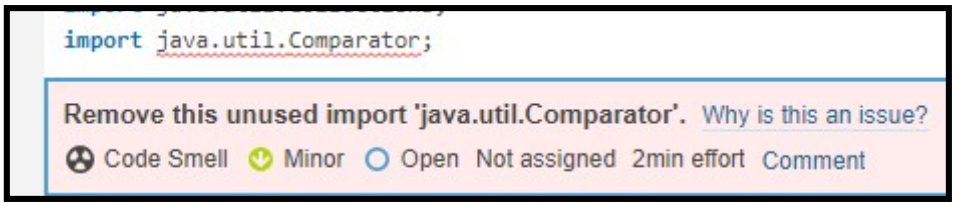


Figura 66. Code smell de categoría "Minor" en la aplicación pública Struts 2.

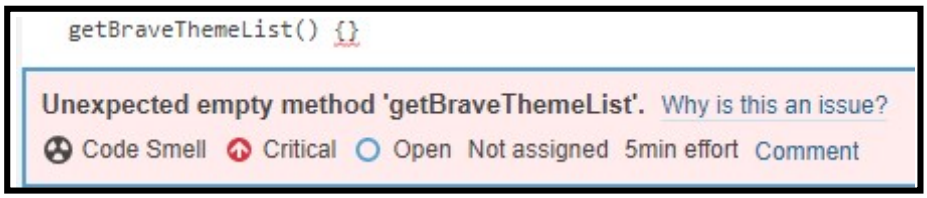


Figura 67. Code smell de categoría "Critical" en la aplicación pública brave-core.

5.3.4 Test (Coverage/Cobertura).

- Condition coverage: Por cada línea con una condición booleana, se comprueba el resultado tanto para el caso verdadero como el falso. De esta manera, se tiene control de la ejecución de las pruebas unitarias.

$$Condition\ Coverage = \frac{(CT\ +\ CF)}{2 * B}$$

  - CT: condicionales que se han probado en el resultado verdadero.
  - CF: condicionales que se han probado en el resultado falso.
  - B: total de condicionales.

- Conditional coverage hits: lista de condicionales probados.
- Conditions by line: condicionales por línea.
- Covered conditions by line: Número de condicionales probados por línea.
- Line Coverage: comprueba si cada línea es probada por las pruebas unitarias.

$$Line\ Coverage = \frac{LC}{EL}$$

  - LC: líneas probadas
  - EL: total de líneas a probar.

- Coverage: Mezcla entre Line Coverage y Condition Coverage. Se comprueba la cantidad de código que han cubierto las pruebas unitarias.

$$Coverage = \frac{(CT\ +\ CF\ +\ LC)}{2 * B + EL}$$

  - CT: condicionales que se han probado en el resultado verdadero.
  - CF: condicionales que se han probado en el resultado falso.
  - LC: líneas probadas.
  - B: total de condicionales.
  - EL: total de líneas a probar.

- Line coverage hits: Líneas de código probadas.
- Skipped unit tests: número de pruebas unitarias sin realizar.
- Uncovered conditions: número de condicionales que no están cubiertos por las pruebas unitarias.
- Uncovered lines: Número de líneas que no están cubiertas por las pruebas unitarias.
- Unit tests: número de pruebas unitarias.
- Unit test duration: Tiempo que ha sido necesario para realizar las pruebas unitarias.
- Unit test errores: número de pruebas unitarias fallidas.
- Unit test failures: Número de pruebas unitarias que han tenido un error de excepción en su ejecución.
- Unit test sucess density (%):

$$Test\ sucess\ density = \frac{Unit\ tests - (Unit\ test\ errors + Unit\ test\ failures)}{Unit\ test * 100}$$

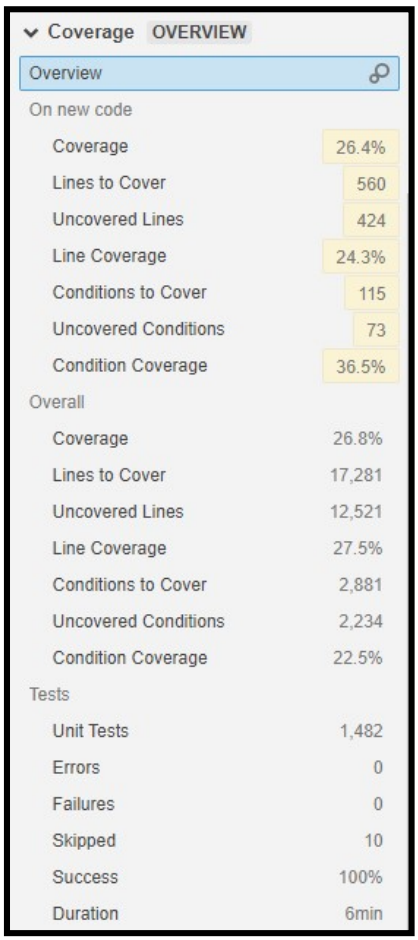


Figura 68. Ejemplo de resultados de prueba en el proyecto blackduck-alert.

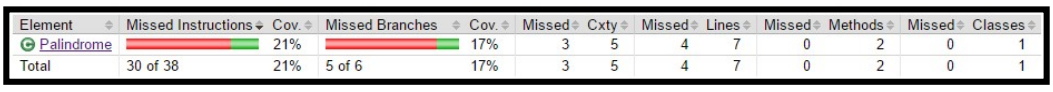
5.3.4.1 Informes de la cobertura.

SonarQube realmente recoge los informes que plugins de diferentes lenguajes utilizan para rellenar las variables que se han mencionado en el apartado anterior.

Por ejemplo, para Java se utiliza JaCoCo, para C# se utiliza dotCover o openCover, PHP utiliza PHPunit. Cada lenguaje de programación utiliza un plugin específico que debe configurarse en la configuración del proyecto en SonarQube o SonarCloud(pestaña Administration).

5.3.4.1.1 Cómo funciona JaCoCo.

JaCoCo es el plugin que más se utiliza en SonarQube y se trata de una biblioteca de cobertura de código gratuita para Java. Para utilizarlo, se debe agregar al proyecto en Java, ya sea desarrollado en Maven o en Gradle, siguiendo manuales que ayudan a añadir los respectivos plugin para su funcionamiento. De esta manera, ejecutando los JUnit test en Eclipse o Netbeans, se realizarán análisis de estas pruebas, mostrando el resultado:



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Palindrome		21%		17%	3	5	4	7	0	2	0	1
Total	30 of 38	21%	5 of 6	17%	3	5	4	7	0	2	0	1

Figura 69. Resultado de JaCoCo en Eclipse sobre un método.

JaCoCo [22] analiza la cobertura de instrucciones, ramas, líneas, métodos, tipos y complejidad ciclomática. No es necesario añadirle archivos fuente y es compatible con todas las versiones de los archivos de clase en java.

La cobertura de instrucciones proporciona información sobre la cantidad de código que se ha ejecutado o se ha perdido.

Las ramas se encuentran en cada *if* y *switch* existentes en el código. Cuenta todas las bifurcaciones en un método y determina el número de bifurcaciones ejecutadas o perdidas. Sin embargo, las excepciones no se toman como bifurcaciones.

Sobre las líneas de código, se considera una línea ejecutada cuando se ha ejecutado al menos una instrucción asignada a esa línea.

Todos los métodos abstractos contienen al menos una línea de código y se considera que se ha ejecutado si una de sus líneas se ha ejecutado. También cuenta como métodos los constructores e inicializadores.

Por último, las clases se consideran ejecutadas si se ha ejecutado al menos uno de sus métodos.

En las propiedades del plugin de JaCoCo se puede añadir una ruta para los reportes para más tarde utilizar estos resultados en herramientas como SonarQube, utilizando el archivo jacoco.exec.

Para que SonarQube pueda leer los reportes que ha realizado JaCoCo en el proyecto, en la pestaña "Administration > General Settings" en la parte izquierda de la pantalla aparece la opción "JaCoCo", donde se escribirá la ruta de los reportes que haya creado JaCoCo.

Con esto, SonarCloud dejará de mostrar un 0% de cobertura a el resultado real de las pruebas que se hayan realizado con JaCoCo:

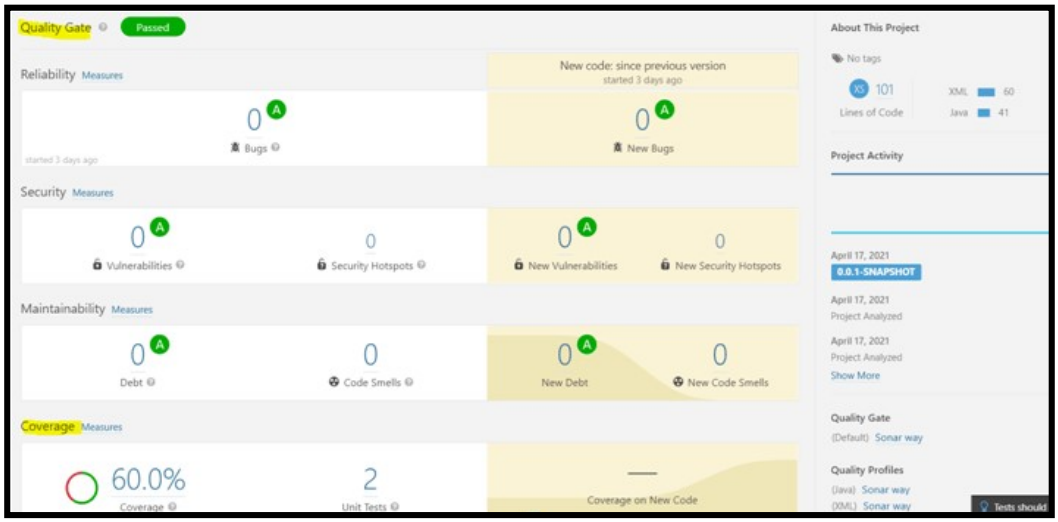


Figura 70. resultado de JaCoCo reflejado en SonarQube.

En el caso de haber una puntuación baja o menor al 100% en la cobertura del código, se deberá crear más pruebas en JUnit que cubran estos errores, de esta manera, el porcentaje de cobertura será mayor y con mejores resultados en el código hasta conseguir un 100%.

5.3.5 Duplications (Duplicaciones).

El número de bloques de líneas duplicadas depende de cada lenguaje. En proyectos que no están desarrollados en Java se siguen las siguientes directices:

- Tiene que haber al menos 100 tokens sucesivos y duplicados.
- Estos tokens deben distribuirse en:
  - 30 líneas en COBOL.
  - 20 líneas en ABAP.
  - 10 líneas en otros lenguajes.

Para los proyectos desarrollados en Java, es suficiente con que haya 10 declaraciones sucesivas y duplicadas, independientemente del número de tokens o líneas. Si hay diferencias de sangría o un string literal, se ignorará en este caso.

- Duplicated files: número de archivos con duplicaciones.
- Duplicated lines: Número de líneas duplicadas.
- Duplicated lines (%):

$$\frac{Líneas\ duplicadas}{LOC * 100}$$

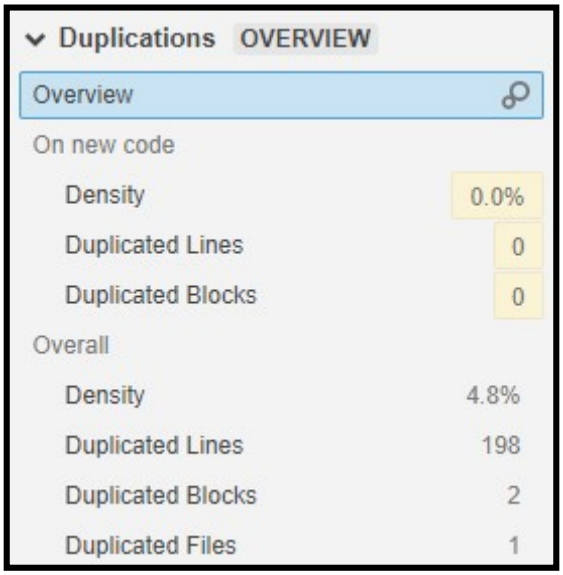


Figura 71. Ejemplo de duplicación de código en el proyecto Teo MotorCortex Player (teo-motorcortex-player).

5.3.6 Size (Tamaño).

- Clases: Número de clases, que incluye clases anidadas, interfaces, enums (clase enumerada con valores constantes) y anotaciones.

- Líneas comentadas: Número de líneas comentadas. No se considerarán líneas comentadas aquellas que solo tengan caracteres especiales, solo se cuentan aquellas con texto. En el siguiente ejemplo se muestran en azul las líneas que no se consideran líneas comentadas y en verde las que sí:

```
/**
*
* public static void main(String args[]){
*       System.out.println("Hola Mundo");
* }
*
* fin
*/
```

  En COBOL, las líneas que contengan las siguientes instrucciones se considerarán tanto líneas de código como líneas comentadas:
  - AUTHOR
  - INSTALLATION
  - DATE-COMPILED
  - DATE-WRITTEN
  - SECURITY

  En Java, los encabezados no se consideran como líneas comentadas, ya que se suelen utilizar para las licencias.

- Comments(%):

$$\frac{Comment\ lines}{(LOC + Comment\ lines) * 100}$$

  - Si el resultado es 50% que el número de líneas de código es igual al número de líneas comentadas.
  - Si el resultado es 100%, todo el archivo está comentado.

- Directories: número de directorios.
- Files: número de archivos.
- Lines: número de líneas físicas (las líneas que se utilizan como tal en el archivo).
- Lines of code: Número de líneas físicas con al menos un carácter que no contiene un espacio en blanco, ni una tabulación ni un comentario.
- En COBOL no se cuentan como líneas de código aquellas que contengan SKIP1, SKIP2, SKIP3, COPY, EJECT o REPLACE.
- Functions: Número de funciones. Dependiendo del lenguaje en el que esté desarrollado el proyecto, se considerará función a una función, un método o un párrafo.
  - En COBOL se consideran el número de párrafos diferentes.
  - En Java los métodos en clases anónimas se ignoran.
  - En Visual Basic .NET los accesors no se consideran métodos.
- Projects: número de proyectos en el Portfolio (disponible en la versión Enterprise de SonarQube).
- Statements: número de declaraciones.

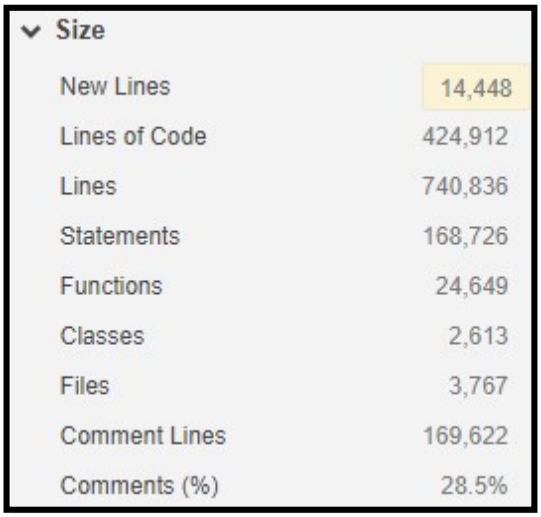


Figura 72. Tamaño del proyecto mediawiki-core.

5.3.7 Issues.

Para ver información más detallada de los posibles problemas, en la barra superior seleccionar "Issues".

- New issues: Número de nuevos asuntos que se han encontrado en un nuevo análisis.
- Tipo de asuntos:
  - Bug.
  - Vulnerability.
  - Code Smell.
- Categoría de los asuntos [23]:
  - Blocker issues: este tipo de problemas pueden afectar a la aplicación en el estado de producción, y deben resolverse inmediatamente.
  - Critical issues: son problemas que pueden dar errores futuros en la aplicación cuando sea desplegada en producción u ocasionar defectos de seguridad. Estos asuntos deben revisarse enseguida.
  - Major issues: Estos asuntos pueden afectar al desarrollador en un futuro, por ejemplo, a la hora de hacer cambios en el código, debido a duplicaciones, parámetros no utilizados…
  - Minor issues: Estos asuntos tienen el mismo efecto que los Major issues, solo que no tienen tanta importancia. Pueden tratarse de líneas muy largas, los switch-cases no deben ser muy largo (3 casos como mucho) …
  - Info issues: No se tratan de problemas o errores, son solo avisos.
- Issues: número de asuntos totales.
- Número de asuntos por cada tipo.
- False positive issues: Número total de asuntos marcados como "Falso positivo". Estos asuntos pueden ser un problema a ojos de SonarQube, sin embargo, el desarrollador puede marcarlo como "False Positive" ya que no lo encuentra como un problema real en el código y puede tener una justificación que SonarQube no vea.

El ciclo de vida de un asunto debe ser el siguiente:

- Open issues: Número de asuntos abiertos. Los asuntos abiertos los abre SonarQube cada vez que hay código nuevo y se realiza un análisis nuevo.
- Confirmed issues: Número de asuntos confirmados. Este estado se cambia manualmente para señalar que el asunto es válido para su inspección.

- Resolved: Este estado se cambia manualmente por el usuario e indica que para el próximo análisis este asunto se debería cerrar.
- Reopened issues: Número de asuntos reabiertos. Este estado es cambiado por SonarQube cuando un asunto que ya estaba resuelto realmente no era correcto y vuelve a dar problemas.
- Closed: Este estado también es modificado por SonarQube cuando los asuntos resueltos han sido correctos al realizar un nuevo análisis al proyecto.

5.3.7.1 Cómo gestionar issues como usuario.

Desde la pestaña Issues ("asuntos" a partir de ahora) en el menú principal del proyecto, se mostrará un resumen de lo encontrado en el proyecto, tanto nuevo como antiguo sin resolver:

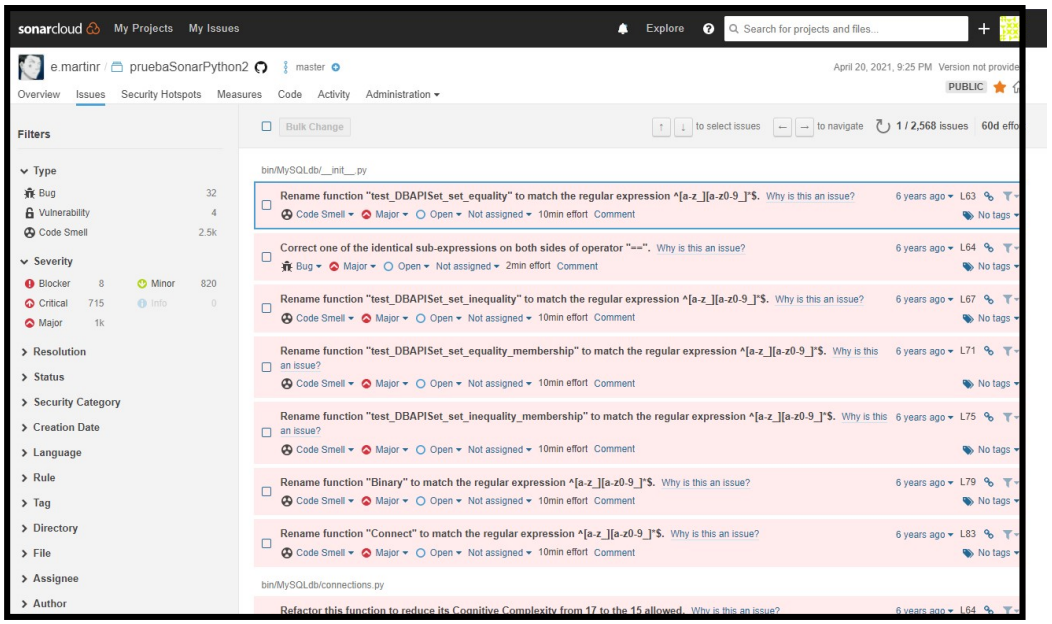


Figura 73. Vista general de los asuntos en un proyecto.

A partir de aquí, se puede filtrar los asuntos en tipo, categoría, estado… entre otros.
A un asunto se le puede cambiar el tipo de esta manera:

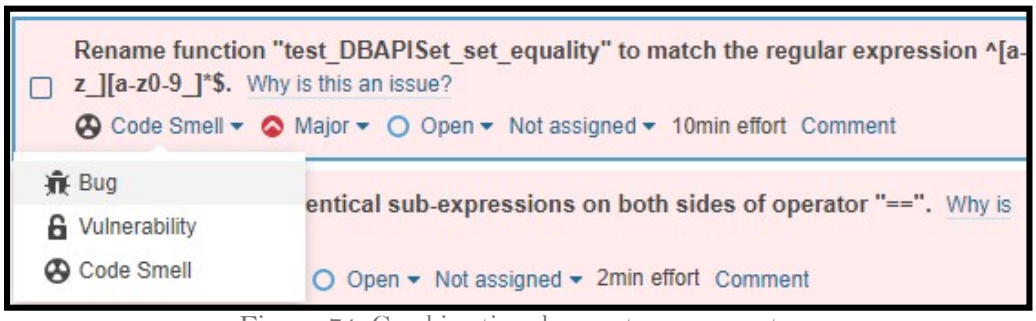


Figura 74. Cambiar tipo de asunto a un asunto.

Igualmente, para la categoría.

Los asuntos se pueden asignar a un usuario, en el caso de este proyecto, lo asigno a mi usuario:

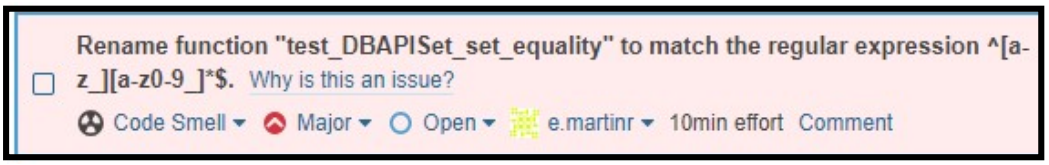


Figura 75. Asignación de usuario a un asunto.

Los asuntos pueden marcarse como abiertos, confirmados, arreglados, como falso positivo o sin resolver:

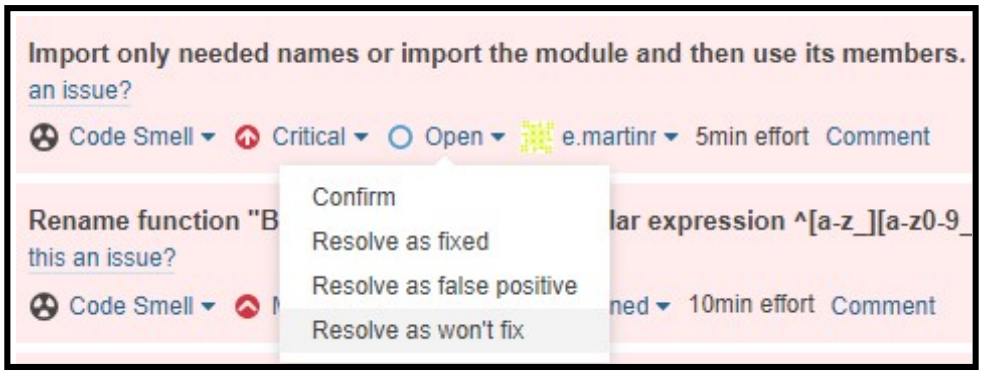


Figura 76. opciones disponibles para cambiar el estado de un asunto.

Si un asunto ha sido resuelto, puede volver a abrirse y cambiará a un estado de re-abierto:

Figura 77. Cambio del estado del asunto de "fixed" a "reopened".

5.3.7.2 Añadir miembros en SonarCloud.

Para añadir un miembro, esta persona debe estar registrada en SonarCloud. El dueño de los proyectos desde su pantalla de usuario selecciona la pestaña “Members” y agrega a un usuario en “Add a member” y lo busca por su nick en SonarCloud.

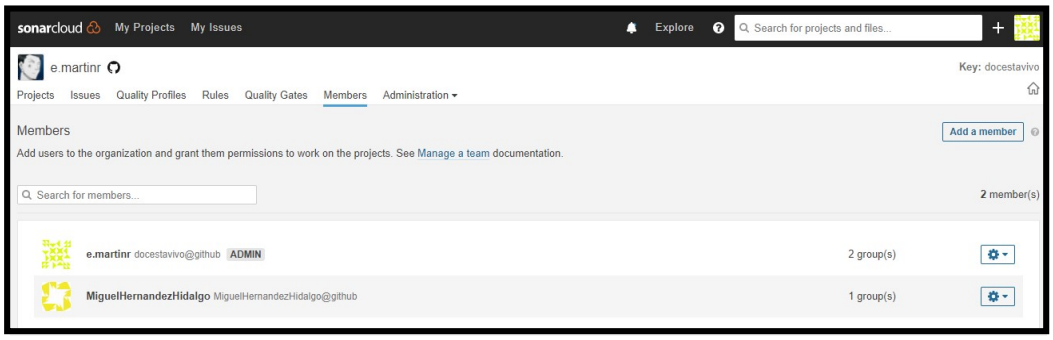


Figura 78. Vista de miembros en SonarCloud.

De esta manera, los nuevos usuarios agregados pueden asignarse asuntos para encargarse de ellos.

Figura 79. Asuntos asignados a usuarios diferentes.

Los usuarios pueden asignarse asuntos entre ellos y cambiar los estados de éstos, no es necesario que lo haga exclusivamente el dueño del proyecto.

Los permisos de los usuarios pueden modificarse en la pestaña de “Administration” en el proyecto en cuestión, ahí se mostrará una tabla con los permisos de los miembros y de los dueños y en la parte inferior, los permisos individuales de cada usuario.

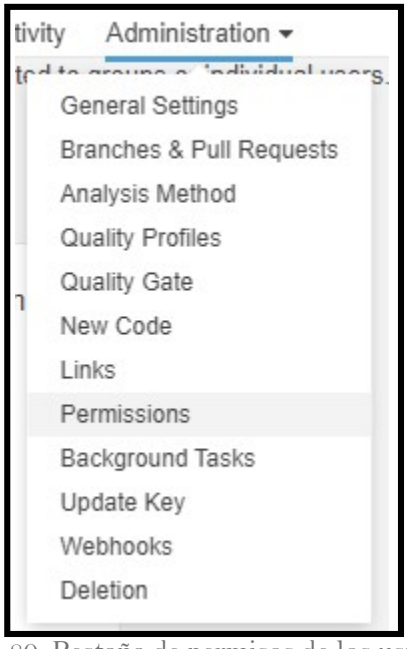


Figura 80. Pestaña de permisos de los usuarios.

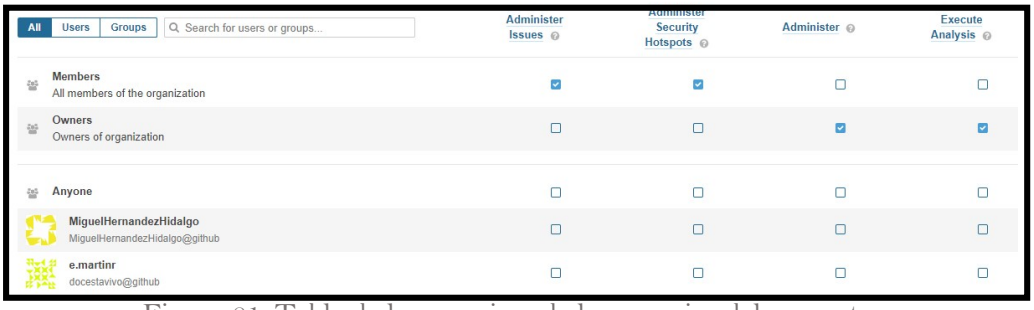


Figura 81. Tabla de los permisos de los usuarios del proyecto.

5.4 Posibles errores en el workflow.

Estos errores se encontrarán en el workflow en GitHub Actions.

5.4.1 Could not find a default Branch to fall back on (común en todos los lenguajes de programación).

Esto es debido a que la Key del proyecto en SonarCloud no coincide con el nombre del repositorio de GitHub. Para solucionarlo, en SonarCloud, en la parte superior seleccionar "Administration" y después "Update Key" y modificarla para que se llame igual que el repositorio en GitHub.

5.4.2 Provide compiled classses with sonar.java.binaries property.

Esto es debido a que en el archivo "sonar-project.properties" en GitHub no contiene la siguiente línea de código: "sonar.java.binaries=RUTA_DEL_CODIGO", donde se debe escribir las carpetas que contengan código Java, separadas con comas en el caso de ser más de una. [24]

Además, en Administration > General Settings > Languages > Java deben estar señaladas las extensiones existentes de los archivos con código (.java, .jav…). Si no se mencionan, el análisis no analizará estos archivos.

5.4.3 Posible caracteres especiales presentes en los archivos.

En el caso de haber un archivo de código a analizar con un carácter especial en el titulo como puede ser la Ñ, hay que modificarlo en el repositorio o el workflow no podrá seguir con sus tareas.

Si hubiera caracteres especiales dentro de los archivos de código, estos se mostrarán como code smell en el futuro análisis, lo mejor es modificarlos y usar los menos posibles.

5.5 Conclusiones.

Es una herramienta muy útil y cómoda para cualquier tipo de proyecto, ya sea grande o pequeño. El único inconveniente que se ha visto es la dificultad que tiene para su instalación cuando se trata de un lenguaje que no permite el análisis automático, ya que antes de la instalación del repositorio desarrollado en java se hicieron otras 3 pruebas antes:

- Creando una máquina virtual con un servidor en el portal de Azure. Al entrar al DNS de la máquina virtual a través del escritorio remoto de Windows, las credenciales no eran correctas cuando sí lo eran. Esto no permitía entrar nunca a la máquina virtual de Windows Server y se descartó.

- También se probó antes de GitHub, Azure DevOps, pero debido a la alta demanda europea en Azure Pipelines, los workflows nunca encontraban un agente y nunca se realizaba la petición a SonarCloud.

- Cuando se empezó a hacer pruebas con GitHub, se comprobó que los proyectos desarrollados en C++ no siguen el mismo manual que puede seguir, por ejemplo, un proyecto desarrollado en Java. Para C++ se requieren otros pasos más complicados que no cumplían con la simpleza que quiere seguir este documento a la hora de la instalación y utilización de las herramientas que se han mencionado a lo largo de este trabajo. También se recomendaba utilizar otras aplicaciones de SonarQube como la extensión que tiene disponible para Visual Studio que analiza el código en tiempo real llamada Sonarlint.

- A la hora de instalar el repositorio de GitHub desarrollado en Java en SonarCloud, se notó que los pasos que se describen por parte de SonarCloud son muy escuetos y no contemplan ciertos aspectos como los que se describen de la Key del proyecto, las acciones para nuevo código o las configuraciones generales por lenguaje que puedan existir. Se tuvieron ciertos problemas la primera vez que se instaló y esto se considera ciertamente molesto para alguien nuevo en esta herramienta que esté utilizándolo sin conocimiento, ya que no existen manuales más minuciosos por cada lenguaje o CI que puede ser utilizado en SonarCloud. Por ejemplo, para Azure DevOps se encontró un manual oficial de AzureDevOps [25] muy detallado de los pasos a seguir para poner en marcha SonarCloud, sin embargo, para GitHub no se encontró ninguno, en este caso, hubo que visitar foros con personas que compartían los mismos problemas y que eran resueltos por soporte de SonarQube.

Sin tener en cuenta los aspectos de la instalación, la herramienta contiene una amplia documentación sobre todos los aspectos que se puedan descubrir sobre los resultados que puede mostrar a cerca de los análisis que haya hecho. Ayuda a los desarrolladores a ver tanto gráficamente como en el propio código los problemas que pueden existen dentro de su proyecto y es una herramienta muy completa para llevar un mantenimiento futuro del programa y pruebas unitarias.

Capítulo 6. Conclusiones y Trabajos futuros

6.1 Conclusiones del TFG

Todas las herramientas a excepción de SonarQube me han resultado muy fáciles de utilizar y considero que pueden resultar muy atractivas para cualquier estudiante que necesite apoyo en sus tareas. Ayudan a comprender el código con el que se está tratando y también a entender conceptos técnicos sobre el software. Hay muchas herramientas disponibles en Internet para muchos lenguajes de programación, las hay que abarcan varios y otras solo especializadas en uno de ellos. Sobre todo, he encontrado muchísimas para lenguajes utilizados en programación web, pero considero que la mayoría de ellas cumplen con las tareas que prometen a pesar de poder estar capadas en las opciones gratuitas que se han analizado.

Sin embargo, SonarQube la posiciono en un lugar especial ya que trata de abarcar casos mucho más prácticos y amplios que requiera que el usuario ya tenga nociones de las medidas de la calidad del software y de programación: serán necesarias para poder interpretar lo que SonarQube pueda estar mostrando. Es una herramienta difícil de instalar, utilizar e interpretar, su documentación es amplía, pero en ocasiones es escueta en sus explicaciones y si se insiste en los foros con el personal de la empresa de ciertos temas que requieren más detalles, por ejemplo, cómo se justifican exactamente los tiempos de la deuda técnica que se muestran en los *issues* de la herramienta, es muy posible que dejen el tema. Considero que siendo un producto tan complejo como es SonarQube, que lo utilizan tanto usuarios como empresas, debería tener una atención al cliente y una documentación mejor, ya que muchos de los manuales que se encuentran y que realmente ayudan a entender el proceso de instalación son precisamente externos a las paginas oficiales de SonarQube. Por otro lado, olvidando todo esto, considero que es una herramienta que una vez entendido cómo es su funcionamiento, puede ayudar a ver a una escala más grande resultados relacionados con la calidad del software. Y a la hora de llevar un proyecto real, es una herramienta que facilita mucho la planificación y la resolución a problemas existentes en el código.

En definitiva, todas estas herramientas van a ayudar a comprender, de manera gráfica y práctica, muchos conceptos que pueden resultar tediosos para una persona que está empezando a introducirse en la ingeniería del software y principalmente en la calidad del software.

6.2 Trabajos futuros

Se plantea la utilización de la herramienta SonarQube de una manera más profesional, con intención de optar por una de sus cuotas para empresas y utilizarlo en una aplicación desarrollada en múltiples lenguajes y entornos. De esta manera ya no se utilizará SonarCloud si no su propio entorno sobre un servidor en la red y herramientas como SonarLint en Visual Studio.

Se quiere mostrar esta vez la integración como diferentes programas y lenguajes y profundizar en el estudio sobre la deuda técnica calculada por SQUALE cuyo plugin solo se encuentra disponible para empresas.

Para todo esto, se planteará una aplicación web ya desarrollada por la empresa A. que entra en la fase de pruebas y análisis del código para más tarde desplegar el programa a producción y

empezar en la fase de mantenimiento. Se tratará de una aplicación orientada a la gestión de emergencias de una comunidad autónoma, lo cual conlleva integraciones con otros servicios de atención a emergencias externos al cliente principal y gran cantidad de pruebas unitarias necesarias para que la aplicación funcione con el menor número de errores y/o fallos posibles en su ejecución dada la gravedad de los asuntos que pueden llevarse a cabo en esta aplicación.

Con esto se plantea demostrar si la herramienta SonarQube es óptima para proyectos de esta envergadura, donde diferentes usuarios puedan realizar análisis del código y los resultados de éstos sean útiles a la hora de mejorar o ajustar un proyecto como el que se propone.

Referencias

[1] "Ingeniería inversa", *Es.wikipedia.org*, 2021. [Online]. Disponible en: https://es.wikipedia.org/wiki/Ingenier%C3%ADa_inversa. [Visitado: 09- Ene- 2021].

[2] Rubén Garrote García, *Reversing. Ingeniería inversa: teoría y aplicación*. RA-MA, 2017, pp. 16-20.

[3] M. Sicilia, *Técnicas de Mantenimiento de Software*. 2009, capítulo 5.

[4] "DBML - Full Syntax Docs | Table definition", *Dbml.org*, 2020. [Online]. Disponible en: https://www.dbml.org/docs/#table-definition. [Visitado: 20- Dic- 2020].

[5] "DBML - Full Syntax Docs | Column settings", Dbml.org, 2020. [Online]. Disponible en: https://www.dbml.org/docs/#column-settings. [Visitado: 20- Dic- 2020].

[6] "DBML - Full Syntax Docs | Default value", Dbml.org, 2020. [Online]. Disponible en: https://www.dbml.org/docs/#default-value. [Visitado: 20- Dic- 2020].

[7] "Use The SQL Plugin To Create An Entity Relationship Diagram", draw.io, 2021. [Online] Disponible en at: https://desk.draw.io/support/solutions/articles/16000082007 [Visitado: 1-Ene-2021].

[8] Myler, H., 1998. *Fundamentals of engineering programming with C and Fortran*. Cambridge: Cambridge University Press, pp.32 - 34.

[9] Es.wikipedia.org. 2021. *Diagrama de flujo*. [online] Available at: <https://es.wikipedia.org/wiki/Diagrama_de_flujo#cite_note-Myler1998-4> [Accessed 17 Febrero 2021].

[10] Shelly, G., Vermaat, M., Quasney, J., Sebok, S. and Freund, S., 2012. *Discovering computers*. Boston, Mass.: Course Technology/Cengage Learning, p.691.

[11] Luis Fernández Sanz, Tema 4. Medición del software. 2019, pp. 5, 20, 59, 60, 61, 66.

[12] Jyoti J. Malhotra y Bhavana S. Tiple, Software Testing and Quality Assurance. Nirali Prakashan, 2011, pp. 3.43-3.45.

[13] Linda M. Laird and M. Carol Brennan, Software Measurement and Estimation: A practical Approach. John Wiley & Sons, 2006, pp. 63, 64.

[14] "SonarQube Documentation", Docs.sonarqube.org, 2021. [Online]. Available: https://docs.sonarqube.org/latest/. [Accessed: 14- Apr- 2021].

[15] "Metric Definitions | SonarQube Docs", Docs.sonarqube.org, 2021. [Online]. Available: https://docs.sonarqube.org/latest/user-guide/metric-definitions/. [Accessed: 20- Apr- 2021].

[16] "Concepts | SonarQube Docs", Docs.sonarqube.org, 2021. [Online]. Available: https://docs.sonarqube.org/latest/user-guide/concepts/. [Accessed: 20- Apr- 2021].

[17] "Release Criteria - The Document Foundation Wiki", Wiki.documentfoundation.org, 2021. [Online]. Available:

https://wiki.documentfoundation.org/Release_Criteria#Blocker_Bug_Definition. [Accessed: 20- Apr- 2021].

[18] "Security Hotspots | SonarQube Docs", Docs.sonarqube.org, 2021. [Online]. Available: https://docs.sonarqube.org/latest/user-guide/security-hotspots/. [Accessed: 20- Apr- 2021].

[19] D. Guamán, P. Quezada-Sarmiento, L. Barba-Guaman and L. Enciso, *Uso de SQUALE y herramientas para análisis e identificación de deuda técnica de código a través de análisis estático*. Lisboa, Portugal: IEEE, 2017.

[20] Es.wikipedia.org. 2021. *Transferencia de Estado Representacional - Wikipedia, la enciclopedia libre*. [online] Available at: <https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional> [Accessed 26 June 2021].

[21] Es.wikipedia.org. 2021. *Facade (patrón de diseño) - Wikipedia, la enciclopedia libre*. [online] Available at: <https://es.wikipedia.org/wiki/Facade_(patr%C3%B3n_de_dise%C3%B1o)> [Accessed 26 June 2021].

[22] Jacoco.org. 2021. *JaCoCo - Documentation*. [online] Available at: <https://www.jacoco.org/jacoco/trunk/doc/index.html> [Accessed 26 June 2021].

[23] "Issues | SonarQube Docs", Docs.sonarqube.org, 2021. [Online]. Available: https://docs.sonarqube.org/latest/user-guide/issues/. [Accessed: 21- Apr- 2021].

[24] "Java | SonarQube Docs", Docs.sonarqube.org, 2021. [Online]. Available: https://docs.sonarqube.org/latest/analysis/languages/java/. [Accessed: 19- Apr- 2021].

[25] "Driving continuous quality of your code with SonarCloud", *Azuredevopslabs.com*, 2021. [Online]. Available: https://azuredevopslabs.com/labs/vstsextend/sonarcloud/. [Accessed: 21- Apr- 2021].

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá