Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Informática

Aplicación UMLConverter

UMLConverter App

Adrián Epifanio Rodríguez Hernández
alu0101158280@ull.edu.es

La Laguna, 10 de marzo de 2023

D. **Jesús Alberto González Martínez**, con N.I.F. 43779378M profesor Colaborador de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

"Aplicación UMLConverter"

ha sido realizada bajo su dirección por D. **Adrián Epifanio Rodríguez Hernández**,

con N.I.F. 78766876H

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 10 de marzo de 2023

Agradecimientos

En primer lugar, deseo expresar mi agradecimiento a todo el profesorado que me ha estado impartiendo clases y motivando a lo largo del grado. Gracias por todo lo que me han enseñado y gracias por todos los conocimientos que he podido adquirir durante sus clases. Gracias a ellos he sido capaz de realizar este trabajo de fin de grado.

Por otro lado, quiero agradecer especialmente a mi tutor, Jesús Alberto González Martínez, por todo el tiempo dedicado, el apoyo e interés mostrados han sido muy importantes para ayudarme a mantenerme motivado durante el transcurso de este trabajo. Gracias por aportarme ideas sobre cómo poder mejorar este proyecto y por hacerme ver la utilidad de los diagramas UML durante las clases en *Fundamentos de la Ingeniería del Software.* Gracias por su ayuda en la planificación, información y organización de este trabajo de fin de grado.

También, quiero agradecer a la Universidad de la Laguna por acogerme en sus aulas, en las que he conocido a personas increíbles que me han apoyado y ayudado a ver las cosas desde distintos puntos de vista. Gracias a el Dr. D. Casiano Rodríguez León por enseñarme las utilidades y funciones de los árboles sintácticos y facilitarme así a enfocar el diseño de este TFG. Gracias a mis amigos y compañeros de la universidad, que siempre me han prestado un gran apoyo moral y humano, animándome a seguir motivado en días duros.

Pero, sobre todo, gracias a mis padres y a mi hermano, por su paciencia, comprensión, apoyo y facilitarme las cosas para poder dedicarme completamente a la realización de este trabajo de fin de grado.

Desarrollar este trabajo ha tenido un gran impacto en mi persona, y es por eso que me gustaría agradecer a todas aquellas personas que me han apoyado a lo largo de este proceso. Sin duda, ha sido un periodo de aprendizaje científico y personal.

A todos, muchas gracias.

Licencia

Resumen

El objetivo de este trabajo de fin de grado ha sido el desarrollo de una aplicación para la generación de diagramas de clases UML sobre proyectos ya existentes. Para ello, UMLConverter es capaz de leer el código fuente de una aplicación que le será proporcionada y genera un diagrama de clases correspondiente a la misma. Además, el programa realiza una pequeña evaluación del software indicando los niveles de calidad del código de la misma.

UMLConverter es capaz de funcionar con programas en *Python* y en *JavaScript*, no se descarta la idea de seguir añadiendo la posibilidad de trabajar con más lenguajes en un futuro. Con este objetivo, he implementado UMLConverter de tal forma que es bastante sencillo generar un nuevo módulo para el lenguaje deseado y añadirlo sin tener la necesidad de cambiar prácticamente nada del programa actual.

Para el desarrollo de este TFG he empezado con la preparación del proyecto, en la que he generado un calendario con los plazos para cada una de las tareas. También he realizado una búsqueda de proyectos similares y he generado una pequeña muestra con pequeños programas para probar su funcionamiento.

En lo relacionado a los resultados de la ejecución del programa, no solo se genera una imagen con el diagrama, sino que se genera un informe en formato Markdown. En este informe aparece el diagrama de clases correspondiente con el tema deseado aplicado, también aparece una tabla con métricas para la evaluación del código de cada clase y una tabla de las métricas correspondiente a cada paquete. Por último, en este informe también aparece la definición de las métricas empleadas.

Palabras clave: UMLConverter, diagrama de clases, UML, Python, clase, métricas, Markdown

Abstract

The objective of this Thesis has been the development of an application for the generation of UML's class diagrams on existing projects. For be able to do it, UMLConverter is able to read the source code of a given application and generate the corresponding class diagram. Moreover, the program makes a small software quality evaluation and shows to the user some metrics with the quality level of the program.

UMLConverter is able to work with Python programs and JavaScript programs. There still the possibility of upgrade the program to allow it to work with more programming languages in the future. The program has been implemented on such a way that it is easy to develop and add a new module with the new language without needing to change almost anything of the actual program.

To the development of this Thesis, I have started with the project preparation, in which I have generated a calendar with all the tasks and deadlines assigned to each one. Furthermore, I have searched about similar projects and I have generated some sample programs for trying the functionalities of this project.

On the other hand, in the related with the UMLConverter execution results, it does not only generate an image with the class diagram, but also generates a Markdown report. That report shows the image with the legend of the selected theme. In addition, it shows some tables with the executed program metrics, one table with each class metric and one table with each package metric. At the end of the report, there is an explanation of each metric and how is the score of each class and package calculated.

Keywords: UMLConverter, class diagram, UML, Python, class, metrics, Markdown

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1 Introducción

El proyecto se define como una aplicación capaz de generar un diagrama de clases UML básico a partir del código de una aplicación ya creada. Para ello UMLConverter es capaz de leer el código fuente de la aplicación que le será proporcionado y genera una estructura UML con las clases y relaciones que existen entre estas.

Por el momento, UMLConverter es capaz de funcionar para programas escritos en *Python* [1] y *JavaScript*. No se descarta la idea de implementar nuevos módulos para añadir la posibilidad de que funcione con más lenguajes. Por ello, se ha desarrollado de tal forma que facilite esta posibilidad.

Entre los principales motivos para la elección de este proyecto se encuentran la ausencia de herramientas que realicen este trabajo, si bien es cierto que existe algunas aplicaciones que realizan una tarea similar a la propuesta, por lo general están bastante limitadas, ya sea, por solo permitir la transformación de un único lenguaje como es el caso de *Eclipse [2]* con el lenguaje *Java*, en el que además tiene que ser mediante su entorno de desarrollo (IDE), o por ser funciones premium de otros programas.

1.2 Descripción UML

Los diagramas UML (Lenguaje Unificado de Modelado) [3] están consolidados como los diagramas estándar para el análisis y diseño de sistemas informáticos. Mediante estos, es posible visualizar de forma rápida y sencilla la estructura básica y las relaciones de un código fuente o de un programa.

El objetivo de este lenguaje es generar un modelo visual para la arquitectura, diseño e implementación de sistemas de software. Por lo tanto, podría definirse este lenguaje como la equivalente versión para el software de unos planos de construcción.

Para el desarrollo de este TFG, únicamente me he centrado en uno de los distintos tipos de diagramas que permite este lenguaje, en este caso, los diagramas de clases UML. Este tipo de diagrama muestra un modelo con la estructura de clases, atributos, operaciones y relaciones entre las distintas clases de un software.

Estos diagramas, suelen ser empleados por los ingenieros de software para documentar la arquitectura de un determinado software. Gracias a ellos, se permite a los programadores ilustrar modelos de datos sin importar la complejidad de los mismos, comprender mejor la visión general de los esquemas de una aplicación, expresar visualmente las necesidades de un sistema y ofrecer una descripción sobre los tipos de datos empleados en un sistema.

El los diagramas de clases, la descripción de una clase está dividida en tres secciones. En la sección superior se incluye el nombre que identificará a la clase. Las siguientes dos secciones se organizan en forma de lista, es decir cada nueva línea indicará un nuevo elemento de la sección correspondiente. En la sección central se muestran los atributos de la clase con sus respectivas visibilidades (privado, público o protegido). Por último, en la sección inferior se muestran las operaciones o métodos pertenecientes a la clase también con sus respectivas visibilidades.

1.3 Antecedentes y estado actual del tema

Existen diversos tipos de diagramas UML, en este caso la aplicación desarrolla un diagrama de clases sobre el programa utilizado, es decir, se realiza ingeniería inversa sobre un código ya existente. Estos diagramas muestran los miembros de las clases con sus respectivas visibilidades y ámbitos de cada clase. Además, también muestran las relaciones entre las distintas clases definidas en un programa.

En la actualidad no existen muchas opciones de aplicaciones que realicen esta tarea, es por ello que se ha propuesto el desarrollo de UMLConverter. La generación de diagramas de clase ha sido de gran utilidad para ciertas áreas como pueden ser banca donde aún existe mucho software en funcionamiento que fue desarrollado antes de que predominasen los métodos de desarrollo ágiles. Es por esto que dicho software ha sido mantenido y desarrollado sin generar una documentación adecuada.

Actualmente se suele definir primero un diagrama UML previo al desarrollo de un proyecto en lugar de a posteriori. Si bien es cierto que eso es lo ideal, aún existen casos donde este diagrama no existe o no es preciso, por ello la utilización de UMLConverter puede agilizar el trabajo de los desarrolladores realizando esta tarea.

Considero que sería una aplicación de gran utilidad para poder conocer la estructura y relaciones de un programa de forma rápida y sencilla, suponiendo que no disponemos de un UML o algún diagrama equivalente que nos proporcione ya dicha información.

1.3.1 Proyectos Similares

Antes de comenzar a realizar este trabajo, he realizado una búsqueda de información sobre proyectos similares a la propuesta. He encontrado algunos proyectos o aplicaciones que realizan la tarea propuesta o parte de ella.

Para empezar, tenemos *Eclipse [2]*, un entorno de desarrollo para aplicaciones *Java* que permite generar diagramas de clase UML básicos con dicho lenguaje dentro de su entorno de desarrollo. Este programa está limitado ya que únicamente permite generar estos diagramas mediante su IDE y para un único lenguaje de programación.

Otra aplicación similar es *Umbrello Project [4]*, en este caso, funciona para los lenguajes *C, C++, Python, JavaScript* y *Java*. Esta aplicación genera diagramas UML bastante básicos para varios lenguajes, pero carece de opciones de personalización de los diagramas, siendo estos poco visuales. *UMLConverter*, a diferencia de *Umbrello Project [4]*, permite al usuario personalizar no solo el tema del diagrama, sino permite la posibilidad de excluir clases del diagrama y la posibilidad de agrupar en paquetes.

Por último, otro ejemplo de aplicación que realiza funciones similares a este TFG sería *GitUML [5]*, esta aplicación permite la generación de diagramas UML con una tecnología bastante similar a la empleada por *UMLConverter*, pero a diferencia de este TFG, *GitUML [5]* no proporciona al usuario métricas para la evaluación del código o la mantenibilidad del mismo.

Capítulo 2

Preparación

2.1 Plan de Trabajo

Debido a que se tenía intención de presentar este TFG en la convocatoria de marzo, el plan de trabajo ha sido propuesto para 18 semanas, desde la semana del 25 de octubre con la asignación del TFG hasta la semana del 20 de febrero, dejando desde el 27 de febrero hasta las fechas de entrega de memoria y presentación para terminar de editar la memoria y preparar la presentación o como comodín en caso de hubiese surgido algún retraso durante la realización del proyecto (semanas 19, 20 y 21).

Cronograma Plan de Trabajo Inicial

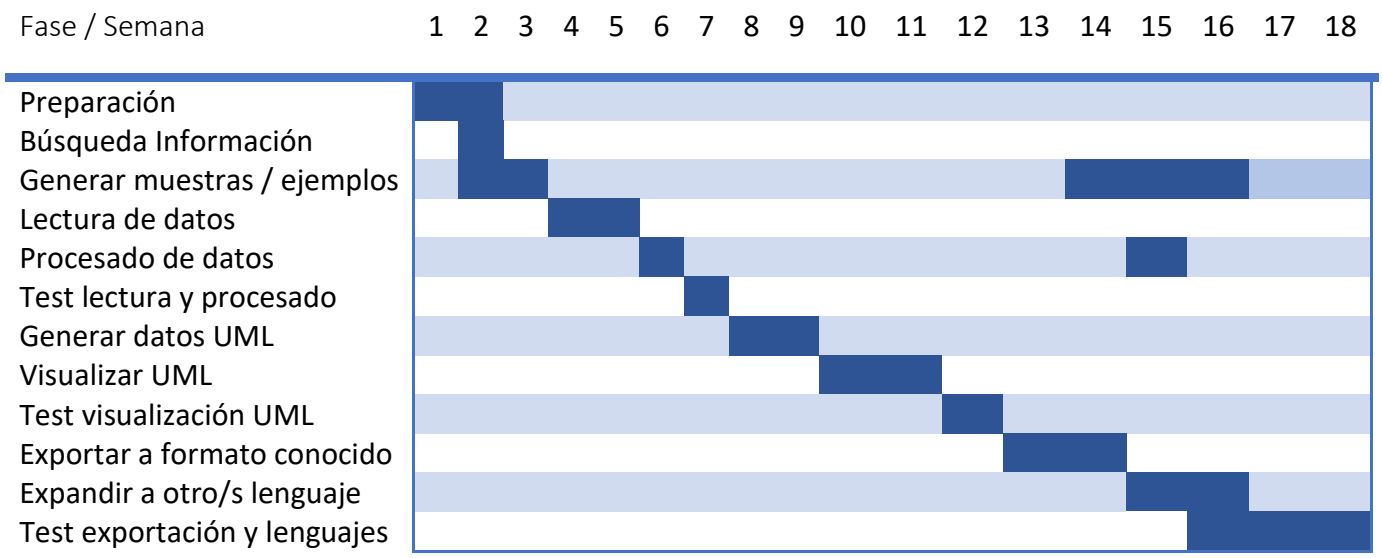

Tabla 1.1: Cronograma con el plan de trabajo inicial para este proyecto.

Una vez comenzado el proyecto y la ejecución del plan de trabajo, me di cuenta de que algunas tareas como la lectura y procesado de datos o la visualización del UML no llevarían tanto trabajo como inicialmente había planificado si se hacía uso de ciertas librerías. Es por esto que, tras hablarlo con el tutor, decidimos que usaría esas librerías que ayudarían a agilizar el trabajo, pero a cambio, incluiríamos una tarea nueva que sería la generación de métricas para evaluar la calidad del código del cual se está realizado el diagrama de clases. Por ello, se añade una nueva tarea para la generación de estas métricas al final del cronograma inicial.

Desglose de semanas

Semana	Fecha Inicio	Fecha Fin
1	25 octubre	30 octubre
2	31 octubre	6 noviembre
3	7 noviembre	13 noviembre
4	14 noviembre	20 noviembre
5	21 noviembre	27 noviembre
6	28 noviembre	4 diciembre
7	5 diciembre	11 diciembre
8	12 diciembre	18 diciembre
9	19 diciembre	25 diciembre
10	26 diciembre	1 enero
11	2 enero	8 enero
12	9 enero	15 enero
13	16 enero	22 enero
14	23 enero	29 enero
15	30 enero	5 febrero
16	6 febrero	12 febrero
17	13 febrero	19 febrero
18	20 febrero	26 febrero
19-21	27 febrero	18 arzo

Tabla 1.2: Desglose con fecha de inicio y de finalización de cada semana empleada en el cronograma.

- **Preparación del proyecto:** Organización para la realización del proyecto, selección de plazos, estructuras de datos, lenguaje a utilizar y entorno de desarrollo, recursos a utilizar…

- **Búsqueda información sobre proyectos similares**

- **Generar muestra con ejemplos de aplicaciones con código fuente y diagrama UML:** Se ha generado una muestra con programas en los distintos lenguajes soportados y sus correspondientes diagramas de clase UML para poder comprobar el correcto funcionamiento de la aplicación.

- **Lectura del código fuente del programa:** La aplicación es capaz de leer correctamente el código fuente del programa en cuestión desde sus correspondientes directorios. Para ello selecciona los archivos con la extensión del lenguaje seleccionado.

- **Procesamiento de los datos:** UMLConverter procesa todo el código leído para únicamente utilizar en el diagrama los ficheros necesarios, seleccionando a su vez de estos el código necesario para la realización del diagrama, eliminando comentarios y porciones de código irrelevantes.

- **Generación de datos para diagrama UML:** La aplicación con los datos procesados genera un archivo con los datos del diagrama de clase UML.

- **Visualización UML:** UMLConverter proporciona una forma de mostrar al usuario el diagrama UML realizado.

- **Exportar a formato conocido:** UMLConverter exporta los resultados del UML a algún formato a *".png"* y *Markdown*.

- **Ampliación a algún otro lenguaje de programación:** Se ha añadido un segundo lenguaje (*JavaScript*) que es soportado por la aplicación.

2.2 Infraestructuras y materiales necesarios

En lo relacionado al material e infraestructuras necesarias para la realización de este trabajo de fin de grado, han sido necesarios un ordenador con python para el desarrollo, conexión a internet para la búsqueda de información y gestión de código mediante *GitHub*, mucho tiempo y ganas.

Por otro lado, los datos para hacer pruebas son importantes, ha sido de gran ayuda disponer de un buen entorno de desarrollo que facilitase las tareas del alumno y disponer de una serie de ejemplos de aplicaciones sencillas con sus respectivos códigos fuente y correspondientes diagramas UML para poder comprobar el correcto funcionamiento de UMLConverter.

2.3 Lenguaje y Estructuras de datos

En un inicio, antes de la aprobación del anteproyecto, me planteé la posibilidad de realizar este TFG con *C#* o *C++,* e incluso comencé a realizar un poco la programación de la estructura básica del programa en ambos lenguajes.

Finalmente, tras estudiar las posibilidades de los distintos lenguajes, me decidí por realizar el proyecto en el lenguaje *Python [1]*. Escogí este lenguaje porque contenía varias librerías que me facilitaban bastante el trabajo a la hora de generar los AST de los que poder sacar luego la información. De haber continuado con *C++* o *C#* hubiese sido necesario un autómata para sacar la información necesaria o generar mi propio *parser* para ello, cosa que hubiese llevado más tiempo y esfuerzo del necesario.

Otro de los motivos por el cual me decanté por esta elección es por el auge de este lenguaje en estos últimos años, y, dado que no lo he dado en profundidad a lo largo

del grado, creí que sería una buena oportunidad para mejorar mis conocimientos en el lenguaje y ganar soltura con el mismo.

Una vez decido el lenguaje, reestructuré el cómo iba a almacenar los datos una vez procesado el árbol, por lo que decidí generar mi propio árbol donde únicamente almacenaría los datos necesarios. Para simplificar la generación, he decidido crear un nodo genérico y un árbol de dichos nodos, de tal forma que implementé la clase *PythonNode* **(Apéndice 1)**. Clase en la que almacenaré el tipo de nodo, el nombre, y valor cuerpo, argumentos si los tuviese. De esta forma, se puede generar un árbol de nodos, ya que en el cuerpo de cada nodo se almacenará un array de nodos.

2.4 Herramientas

Para la realización de este TFG se han empleado diversas herramientas. Para empezar, se ha empleado el entorno de desarrollo *Sublime Text [6]* junto con una terminal de *Linux* para *Windows*. De esta forma he sido capaz de generar el proyecto tanto para *Windows* como para *Linux* y probarlo para ambos sistemas operativos. Además, se ha hecho uso de una máquina virtual para probar el mismo en *MacOS*.

Otra de las herramientas clave para el desarrollo de este proyecto ha sido el uso de la plataforma *GitHub [7],* mediante ella, he podido mantener las distintas versiones del proyecto a la vez que el tutor ha tenido acceso al mismo. Además. *GitHub,* ha permitido la ejecución de las pruebas del proyecto en los distintos sistemas operativos mediante los *Workflows* de la plataforma. Por otro lado, he generado un *Project Board* en la plataforma en el que he ido añadiendo las fases en las que se encontraba el proyecto en cada momento.

Una herramienta clave para el correcto desarrollo de este TFG ha sido el uso del lenguaje *Mermaid [16]* para permitir la generación de diagramas UML. Aunque de momento únicamente empleo este lenguaje en diagramas de clase, existe la posibilidad de emplearlo para otros tipos de diagramas UML que se implementasen para este proyecto en un futuro. Este lenguaje ha facilitado mucho la generación de las imágenes con los diagramas, ya que existen algunas librerías como *PlanteUML* que son capaces de generar los diagramas a partir de código fuente escrito en este lenguaje.

Por último, se han empleado diversas herramientas y librerías para el desarrollo de este TFG, todas ellas relacionadas con el lenguaje empleado (*Python*).

- **Python 3.10**: [1] Ha sido el lenguaje empleado para el correcto desarrollo de este proyecto.

- **VENV**: [8] Es un módulo de python que permite generar un entorno virtual independiente para probar la aplicación. En él se deben descargar todos los paquetes necesarios, pero estos paquetes no generan conflictos con los locales.
- **Pytest**: [9]Es un módulo de *Python* que permite la ejecución y generación de las pruebas de la aplicación, permitiendo de esta forma comprobar el correcto funcionamiento de la misma.
- **Coverage**: [10] Es un módulo de *Python* que permite generar un informe sobre el cubrimiento del código en las pruebas de la aplicación. De esta forma se puede ver de forma rápida y sencilla cuanto porcentaje del código de la aplicación ha sido probado mediante las pruebas de *Pytest.*
- **PlantUML**: [11] Módulo de *Python* que genera la imagen del diagrama de clases UML a partir de un código proporcionado en lenguaje *Mermaid*.
- **Easygui**: [12] Módulo de *Python* que permite la generación de interfaces gráficas de forma sencilla. Este módulo ha sido usado para permitir que la ejecución de la aplicación sea más sencilla y dinámica mediante los menús.
- **Esprima**: [13] Módulo de *Python* que permite la generación de un AST para código en *JavaScript [14]*.

Ejemplo lenguaje Mermaid

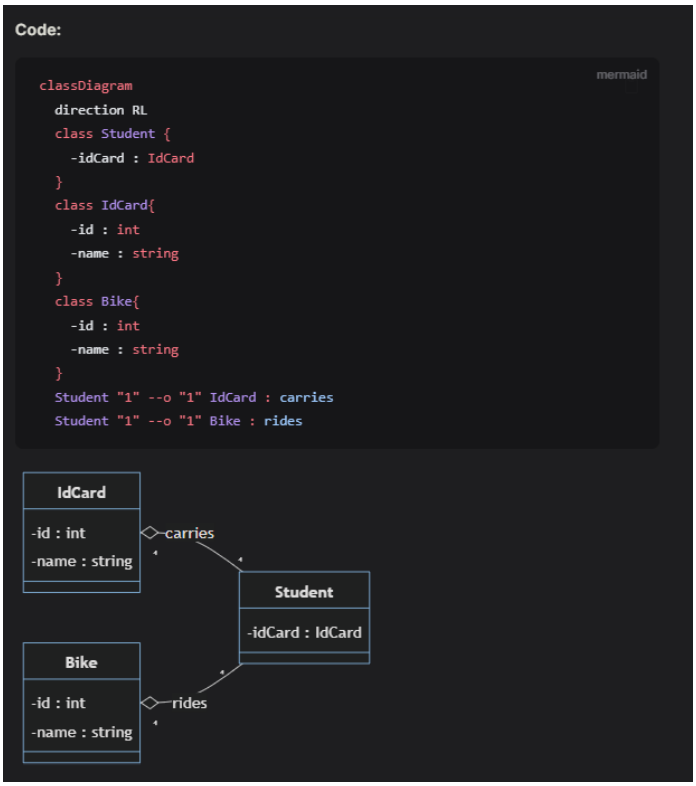

Figura 1: Ejemplo de nuevo árbol a partir del AST procesado.

Capítulo 3

Lectura y Procesado de Datos

3.1 Búsqueda y Selección de Archivos

UMLConverter funciona de momento para proyectos en dos lenguajes (*Python* y *JavaScript*), es por eso que lo primero que tendrá que decidir el usuario al ejecutar esta aplicación es la selección de lenguaje. El lenguaje seleccionado es necesario para que UMLConverter sepa que extensión de archivos debe buscar. Por ejemplo, para el lenguaje *Python* buscará los archivos con extensión *".py"* y para el lenguaje *JavaScript* buscará los archivos con extensión *".js"*.

Por otro lado, el usuario deberá escoger la carpeta raíz del proyecto para que UMLConverter busque los archivos recursivamente dentro de ese directorio. Además, por defecto, excluirá de su búsqueda todos los archivos pertenecientes a algunos directorios, como los que suelen encontrarse en el archivo "*.gitignore"* de un proyecto. Si un usuario desease que se incluyesen los archivos de alguno de esos directorios, dentro del archivo "*app/modules/utis.py*" existe un objeto con el listado de exclusiones que puede cambiar.

Una vez es programa haya encontrado todos los archivos con la extensión pertinente, el usuario tendrá la posibilidad de omitir del diagrama de clases resultante los archivos que desee. Para ello, tiene una opción de personalización en la que puede seleccionar de una lista los archivos que no quiere que se incluyan.

Ejemplo menú de exclusión de archivos

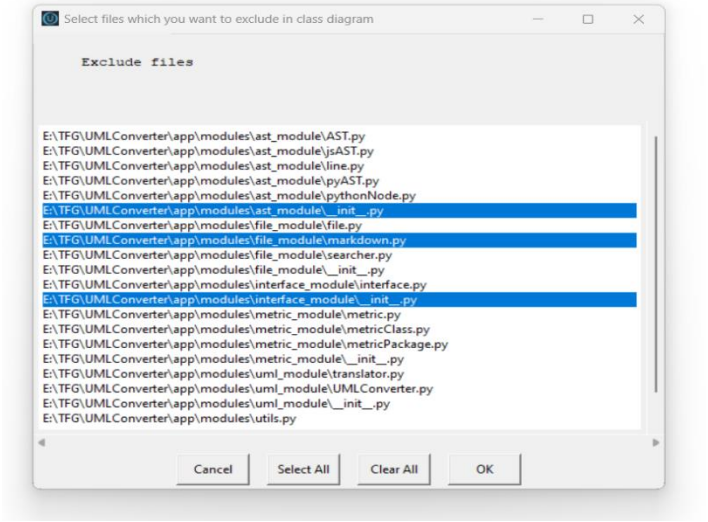

Figura 2: Ejemplo de interfaz con la selección de ficheros que se desean excluir del diagrama.

3.2 Generación y Procesado AST

Para que UMLConverter sea capaz de recopilar la información necesaria para el desarrollo del diagrama de clases hace uso de los árboles sintácticos (AST). Es por esto que una vez el usuario ha finalizado con la fase de búsqueda y selección de archivos, cada uno de estos será enviado a una función para generar su correspondiente AST.

Una vez generados los árboles correspondientes a cada archivo, se procede al procesamiento de los datos, ya que en estos árboles se almacena mucha información que no es necesaria para el desarrollo del diagrama. Para ello, se hace uso de la clase *PythonNode* y se van almacenando en estos nodos únicamente los datos necesarios para el correcto funcionamiento del programa.

Este procesamiento varía en función del lenguaje, para el lenguaje *Python*, se utiliza el módulo *AST [15]* del propio lenguaje. Este módulo que genera el AST no tiene casi documentación sobre los métodos de acceso, es por ello que acceder a los datos del árbol es bastante complejo. Para solucionar este problema he generado un autómata que procesa las líneas del árbol como si fuese un fichero de texto en lugar de un objeto. El programa lee línea a línea y comprueba si el tipo de nodo del AST es necesario para el diagrama de clases, en caso de que lo sea se genera un nuevo nodo *PythonNode* y se almacena en memoria, en caso contrario se avanza hasta la siguiente línea de declaración de nodo.

Para el lenguaje JavaScript, se utiliza el módulo *Esprima [13]* para generar el AST de los ficheros en este lenguaje. Acceder a los atributos de este objeto es bastante sencillo de forma que el procesamiento de los ficheros en este lenguaje es mucho más sencillo que para *Python*. En este caso se van comprobando los tipos de nodos para ver si son redundantes para el diagrama o no, en caso de que el tipo de nodo concuerde con los necesarios se extrae la información y se almacena en un nuevo nodo de tipo *PythonNode.*

Por último, tanto la clase encargada de procesar JavaScript como la encargada de procesar Python, heredan de una clase abstracta definida en el fichero *"app/modules/ast_module/AST.py"*. Si en un futuro se decidiese seguir ampliando los lenguajes para los cuales funciona este programa, sería tan sencillo como implementar un nuevo módulo para el lenguaje correspondiente que contuviese una clase heredada de la clase abstracta y almacenase los datos en nodos *PythonNode*.

Ejemplo de árbol procesado

simpleClass.py

```python
class NewClass():

    var1: str
    var2: int

    def __init__(self, v1, v2):
        self.var1 = v1
        self.var2 = v2

```

simpleClass.txt

```
Module
    Body: [

    ClassDef
        Name: NewClass
        Body: [

        AnnAssign
            Name: var1
            Value: str

        AnnAssign
            Name: var2
            Value: int

        FunctionDef
            Name: __init__
            Args: [
            self
            v1
            v2
            ]
            Body: [

            Assign
                Name: self.var1
                Value: v1

            Assign
                Name: self.var2
                Value: v2
            ]
        ]
    ]
```

Figura 3: Ejemplo de nuevo árbol a partir del AST procesado.

3.3 Almacenamiento de Datos

UMLConverter almacena los datos procesados en objetos de la clase *PythonNode.* Estos nodos van formando una estructura de árbol ya que el cuerpo de estos nodos está formado por un array que contiene los nodos hijo del mismo. De esta forma, es mucho más sencillo acceder posteriormente a la información ya que, a diferencia de en los AST, todos estos nodos tienen la misma estructura.

Almacenar estos nodos en memoria, produce que el programa se ejecute de forma mucho más rápida a si almacenase la información en archivos de texto o se tratase de buscar la información en el AST cada vez que se necesitase.

Por último, gracias al almacenamiento de los datos de esta forma, se permite que este programa pueda funcionar en un futuro para cualquier lenguaje. Se genera cierta abstracción a la hora de añadir módulos para otros lenguajes, esto permite evitar la necesidad de cambiar código en los otros ficheros de este programa.

Capítulo 4

Generación y Visualización

4.1 Transformación a Mermaid

Una de las fases de este proyecto consiste en la generación de datos UML. Para ello, hago uso del lenguaje *Mermaid [16]*. Este lenguaje permite la generación de diagramas UML complejos mediante código evitando la necesidad usar un editor para generar estos diagramas.

UMLConverter emplea su propio traductor para generar de los datos almacenados en memoria un código equivalente en lenguaje *Mermaid*. Para ello, se coge el árbol de nodos *PythonNode* y empieza a buscar recursivamente en él, generando el código correspondiente a cada uno de sus nodos y sus respectivos atributos, cuerpos, parámetros…

En el caso de los nodos de importación, este código no se genera al momento, sino que se guardan las relaciones en una variable hasta que se ha finalizado la traducción del resto del código. Una vez esta traducción ha finalizado para todos los nodos, se mira la lista de importaciones y se incluye la traducción de las importaciones que empleen clases incluidas en el diagrama. En caso de que se importe una clase o un módulo no incluido en el diagrama no se traduce esta importación y se pasa a la siguiente. Esto mismo ocurre para las herencias, en caso de que una clase herede de otra clase no definida en el diagrama, esta herencia no se traduce a *Mermaid*.

Al igual que en muchos lenguajes de programación, *Mermaid* permite definir la visibilidad de los atributos o métodos de las clases (protegido, privado o público). De esta forma se pueden generar diagramas mucho más precisos y visuales. También permite la utilización de distintos tipos de flechas para los distintos diagramas UML, aunque para este TFG únicamente utilizaremos las correspondientes a los diagramas de clases.

Por último, una vez finalizada la traducción tanto de las clases y paquetes como de las importaciones y herencias, estos datos se almacenan en un fichero con extensión *".txt"* que será utilizado posteriormente por el módulo para generar la imagen UML. Al inicio de este fichero también se encontrará el tema seleccionado por el usuario si este hubiese sido cambiado.

Ejemplo de transformación a Mermaid

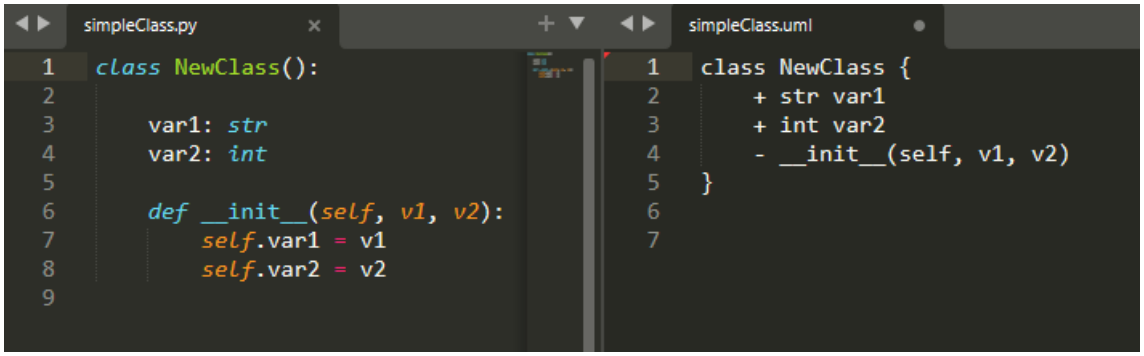

Figura 4: Ejemplo de transformación a lenguaje Mermaid.

4.2 Opciones de personalización con PlantUML

PlantUML [11] es un módulo de *Python* que permite realizar la conversión de código *Mermaid* a una imagen en formato "*.png*". Para ello, es necesario tener instalada previamente la librería. Con la ayuda de este módulo, UMLConverter permite la personalización de varios aspectos del diagrama de clases del resultado.

En primer lugar, el usuario tiene la opción de decidir si quiere que se agrupen las clases del diagrama en sus respectivos paquetes. Por lo que, si el usuario ha habilitado esta opción en el menú, se agruparán las clases en los paquetes leídos por el programa. Los paquetes serán las subcarpetas en las que se encuentren los archivos, por lo que todas las clases definidas en archivos de una misma carpeta pertenecerán al mismo paquete. Ese paquete tendrá como nombre el mismo que el de la carpeta que los contiene.

Ejemplo diagrama clases UML sencillo

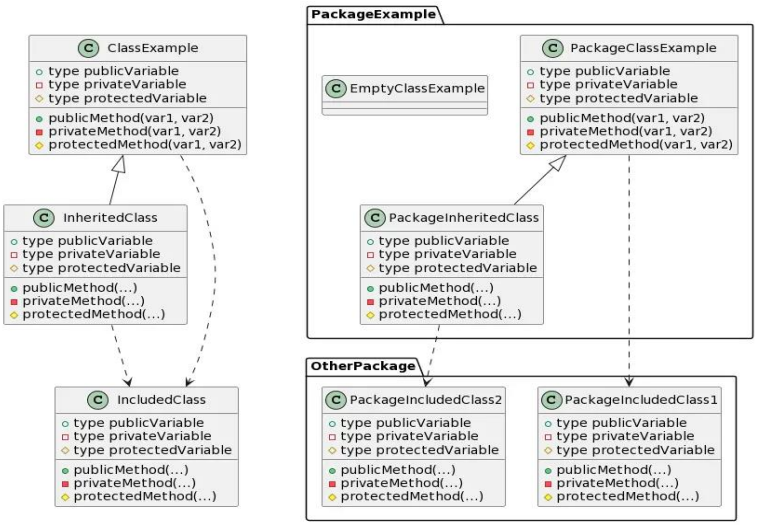

Figura 5: Ejemplo de diagrama sencillo generado con y sin uso de paquetes.

Por otro lado, UMLConverter permite la exclusión de archivos del diagrama final. Es gracias a esto, que, si un usuario desease que ciertas clases o archivos no aparecieran en el diagrama, ya sea para reducir la complejidad de este o por cualquier otro motivo, tiene la opción de seleccionar todos los archivos que desee que sean omitidos desde el menú de la aplicación.

Otra de las opciones de personalización que se le permite al usuario es el tema con el que se generará el diagrama de clases. De esta forma, el usuario tiene una variedad de hasta 15 temas distintos para poder cambiar los colores y formatos del resultado final.

Por último, el usuario puede decidir el peso que tendrá cada una de las métricas en el informe *Markdown* o si desea excluirlas del resultado. Para ello, en el menú de la aplicación deberá rellenar unos campos con el porcentaje que desee que cuente cada una de las métricas al resultado de la evaluación de cada clase y paquete.

Ejemplo selector de temas

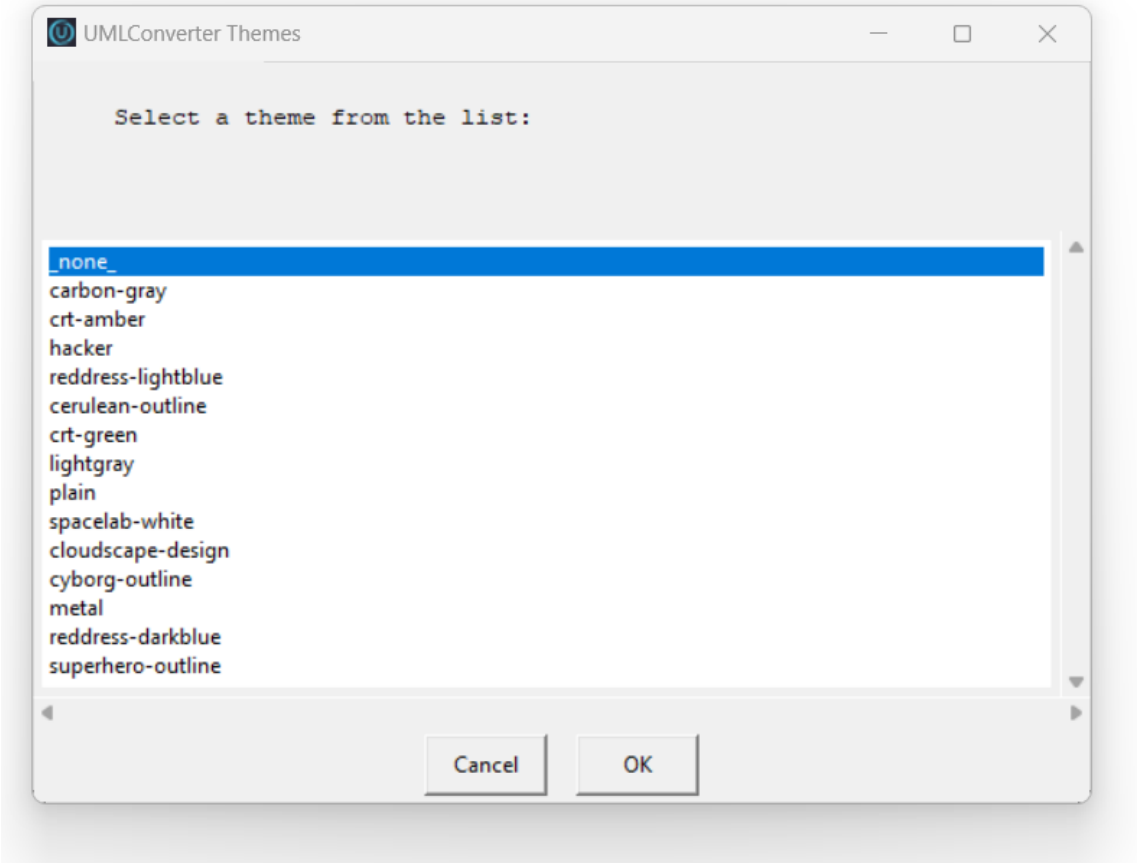

Figura 6: Ejemplo de pantalla de selección de tema para el diagrama de clases.

Capítulo 5

Métricas

5.1 Métricas de Clases

UMLConverter evalúa tres métricas distintas para cada una de las clases en el diagrama. Estas evaluaciones se realizan siguiendo las fórmulas definidas en el artículo de M. A. Sicilia [17]. Estas métricas han sido desarrolladas en una clase llamada *MetricClass* que permitiría poder añadir nuevas métricas en un futuro sin dificultad.

- ***NOC (Number Of Children):*** Cantidad de clases que heredan directamente de esta clase. El número de hijos (NOC) ayuda a conocer el nivel de conexiones entre clases y la cantidad de métodos que pueden ser reutilizados. Un alto NOC implica alto número de reutilización de métodos, pero dificulta la mantenibilidad del código.

- ***CCD (Code Comments Density):*** Es la densidad de comentarios en una clase, se calcula dividiendo la cantidad de líneas de comentarios entre la cantidad de líneas de código. Esta métrica es un indicador para la mantenibilidad del código ya que una buena densidad de comentarios facilita las tareas de mantenimiento, actualización y/o corrección de errores.

- ***CBO (Coupling Between Object classes):*** El acoplamiento entre clases representa el número de clases relacionadas con la clase evaluada. Para calcular esta métrica se comprueba la cantidad de clases que requieren alguno de los métodos de la clase evaluada en su código. Esta métrica es un indicador de la calidad de una estructura orientada a objetos. Si todas las clases tiene un acoplamiento igual a cero, y una sola clase tiene un acoplamiento elevado, esto indica una mala orientación a objetos.

Ejemplo métricas de clases

Class Table

Name	Package	NOC	CodeLines	CommentLines	CCD	CBO	Score
AST	ast_module	2	55	151	2.75	1	87.17%
JsAST	ast_module	0	137	125	0.91	2	81.76%
Line	ast_module	0	46	51	1.11	0	82.75%
PyAST	ast_module	0	348	251	0.72	2	73.68%
PythonNode	ast_module	0	79	104	1.32	0	82.75%

Figura 7: Imagen con ejemplo de tabla con las métricas de las clases.

5.2 Métricas de Paquetes

UMLConverter evalúa dos métricas específicas para paquetes, y además evalúa como tercera métrica de paquetes la media de las métricas de cada una de las clases contenidas en dicho paquete. Las dos evaluaciones específicas se realizan siguiendo las fórmulas definidas en el artículo de M. A. Sicilia [17]. Estas métricas han sido desarrolladas en una clase llamada *MetricPackage* que permitiría poder añadir nuevas métricas en un futuro sin dificultad.

- ***DIT (Depth of Inhetiance Tree):*** La profundidad del árbol de herencia expresa la profundidad máxima encontrada en las herencias dentro de cada paquete, excluyendo las herencias de clases propias del lenguaje. Esto sirve para poder evaluar la mantenibilidad del código, ya que una jerarquía de herencia demasiado profunda complica el mantenimiento.

- ***LCOM (Lack of Cohesion in Methods):*** La carencia de cohesión en los métodos calcula la cantidad de clases en el paquete que depende de clases externas al paquete. Un valor elevado para esta métrica indica que las clases del paquete podrían diseñarse mejor descomponiéndola en más clases distintas

- ***CAS (Class Average Score):*** Esta métrica representa la puntuación media de las evaluaciones de cada una de las clases pertenecientes al paquete.

Ejemplo métricas paquetes

Package Table

Name	Class Ammount	DIT	LCOM	CAS	Score
ast_module	5	1	0.0	81.0%	90.5%
file_module	3	0	0.5	66.0%	83.0%
interface_module	1	0	0.0	83.0%	91.5%

Figura 8: Imagen con ejemplo de tabla con las métricas de los paquetes.

Capítulo 6

Conclusiones y líneas futuras

El Trabajo de Fin de Grado ha consistido en el desarrollo de una aplicación capaz de generar diagramas de clase UML a partir del código fuente de una aplicación ya existente.

Durante la realización del análisis y diseño de la aplicación, podía apreciar que es un proyecto de gran utilidad para desarrolladores. Además, debido a que es un proyecto bastante grande, se ha tomado la decisión de únicamente implementarlo para que funcione con dos lenguajes de programación (*Python* y *JavaScript*).

Durante la elaboración del proyecto he empleado varias herramientas como *GitHub* para agilizar y gestionar el desarrollo del mismo. Esto ha permitido comprobar que todas las nuevas actualizaciones en el proyecto sean compatibles con versiones anteriores ya que los test ejecutados por el *Workflow* lo comprueban.

Por otro lado, en lo relacionado a las tareas y plazos, se han seguido todas las tareas definidas en el anteproyecto, y, en la mayoría de los casos, estas se han encontrado dentro de sus correspondientes plazos. En otras ocasiones, se han producido adelantos que han permitido añadir funcionalidades nuevas al TFG que inicialmente no estaban establecidas.

A lo largo del desarrollo de este TFG, he aplicado los conocimientos adquiridos en el transcurso de las asignaturas del grado de Ingeniería Informática, como puede ser, el análisis de los requisitos, las distintas metodologías de desarrollo o las habilidades necesarias para el correcto desarrollo del software.

Entre las posibles líneas de futuro para este proyecto, se intentará implementar la posibilidad de funcionamiento con más lenguajes de programación orientados a objetos. También se desarrollará una versión web de la aplicación para que los usuarios puedan ejecutarla de forma online sin necesidad de descargar el ejecutable. Por otro lado, también se pretende realizar una ampliación de las métricas permitiendo que se evalúen nuevas métricas para las clases y paquetes.

Una vez el TFG esté completamente terminado se pondrá en producción y comenzará la fase de mantenimiento. Donde, como desarrollador, debo asegurarme de que la aplicación funcione correctamente y realizar los distintos cambios y actualizaciones que vayan siendo requeridos por los usuarios.

Por último, he escogido el nombre de UMLConverter para este proyecto porque en un futuro me gustaría que este se expandiese a más tipos de diagramas UML, y no únicamente al diagrama de clases que se ha desarrollado durante el transcurso de este TFG. De esta forma, la aplicación tendría mayor versatilidad y utilidad para los usuarios.

Capítulo 7

Summary and Conclusions

This Thesis, has consisted in the development of an application capable of generating class UML diagrams from a given source code of an existing application.

Along the development of the analysis and design of the application, I could see that is a very useful project for developers. Moreover, because it is a really large project, the decision of only implement it for two programming languages was taken (*Python* and *JavaScript*).

During the project elaboration I have used different tools such as *GitHub* to speed up and manage the project development. This has made it possible to check that all the new updates of the application are compatible with the previous versions of it by the use of the *Workflows* each time a commit is pushed to the repository.

On the other hand, in relation to the tasks and deadlines, all the tasks defined in the Thesis draft have been followed, and, in most of cases, these tasks have been finished within the corresponding deadline. On other occasions, some tasks have been finished earlier, so that have allowed to develop new functionalities that were initially not established in the Thesis draft.

Throughout the development of this Thesis, I have applied the knowledge acquired in the different subjects taken along this degree in Computer Engineering, such as the analysis of the requirements, the different development methodologies or the necessary skills for the correct development of the software.

Among the possible future lines for this project, an attempt will be made to try the possibility of be able to work with more different object-oriented programming languages. Also, a web version of this application could be developed so that the users can run it online without having to download the launcher. Another improvement to be done is to develop new metrics for the classes and packages.

Once this Thesis is completely finished, it will be moved to production and the maintenance phase will begin. Where, as a developer, I will have to assure that the application works correctly and make the different changes and updates that are requested by the users.

To conclude, I have chosen the name of UMLConverter for this project because in the future I would like it to be expanded to more UML diagrams instead of only class diagrams as the ones I have developed along this Thesis. That would give more versatility to this application and more utilities for the users who choose to use it.

Capítulo 8

Presupuesto

8.1 Presupuesto hardware

Son necesarios los siguientes elementos hardware para el desarrollo de este TFG:

- Ordenador personal: *Acer Nitro 5 AN515-57-75M9*, procesador *Intel Core i7-11800H* de 2.30 GHz, 32 GB *DDR4 3200 MHz* para la RAM, disco duro *SSD* de 2 TB y sistema operativo *Windows 11 Pro*.
    - Utilización: se estima el tiempo de vida útil en 5 años, por lo que su utilización es del 8.5% (260 semanas ≡ 100% → 22 semanas ≡ 8.5%).

- Monitor externo: *Acer V196HQL*.
    - Utilización: se estima el tiempo de vida útil en 6 años, por lo que su utilización es del 7% (312 semanas ≡ 100% → 22 semanas ≡ 7%).

- Conexión a Internet: conexión fibra óptica de hasta 150 Mb con el operador Vodafone.
    - Utilización: se estima una utilización del 20% durante las 22 semanas del desarrollo del TFG.

- Ratón: *MSI Clutch GM30.*

- Teclado: *BlueStork BS-GKB.*

- Cascos con micrófono: *MSI GH20.*

Presupuesto Hardware

Recurso	**Coste (€)**	**Uso (%)**	**Total (€)**
Ordenador personal	2399,00	8,50	203,92
Monitor externo	89,99	7,00	6,30
Conexión a Internet	5,5 meses * 33,40 = 183,7	20,00	36,74
Ratón	32,98	100,00	32,98
Teclado	26,99	100,00	26,99
Cascos con micrófono	39,99	100,00	39,99
		Total:	346,92

Tabla 2: Desglose Presupuesto hardware.

8.2 Presupuesto software

Son necesarios los siguientes elementos software para el desarrollo de este TFG:

- *Python 10, Sublime Text 3, Opera, Adobe Acrobat Reader,* y diferentes herramientas online. Debido a que este software es gratuito no se tendrá en cuenta en este presupuesto.
- *Microsoft Office 365*: licencia mensual, 8,40€ al mes.
- *Adobe Photoshop 2022*: licencia para un único uso mensual.
- *Adobe Premiere Pro 2022*: licencia para un único uso mensual.

Presupuesto Software

Recurso	Coste (€)	Uso (%)	Total (€)
Microsoft Office 365	5,5 meses * 8,40 = 46,20	100,00	46,20
Adobe Photoshop 2022	5,5 meses * 24,19 = 133,05	100,00	133,05
Adobe Premiere Pro 2022	5,5 meses * 24,19 = 133,05	100,00	133,05
		Total:	312,30

Tabla 3: Desglose Presupuesto software.

8.3 Presupuesto Personal

Los costes generados por el personal necesario para llevar a cabo el TFG se calculan teniendo en cuenta que el proyecto ha sido desarrollado por una única persona que asume los roles de administrador del proyecto, analista, documentador, diseñador, programador y tester. Los precios por hora de cada rol han sido estimados en función del salario medio [13] español para dicho rol.

Presupuesto Personal

Recurso	Tiempo (h)	Coste (€/h)	Total (€)
Administrador del proyecto	20	15,13	302,60
Analista	30	16,41	492,30
Documentador	100	16,66	1666,00
Diseñador	40	12,31	492,40
Programador	320	14,62	5.318,40
Tester	100	13,08	1308,00
		Total:	9.579,70

Tabla 4: Desglose Presupuesto personal.

8.4 Presupuesto total

El presupuesto total del proyecto es la suma de cada uno de los presupuestos individuales. El presupuesto total del proyecto es de 10.238,92 €.

Presupuesto Total

	Total (€)
Presupuesto hardware	346,92
Presupuesto software	312,30
Presupuesto personal	9.579,70
Total:	10.238,92

Tabla 5: Desglose Presupuesto total.

Apéndices

Apéndice I

PythonNode

PythonNode es el nodo genérico en el que se almacenan los datos relevantes para el diagrama de clases. A continuación, se describen los atributos de este objeto:

- **nodeType**: Representa el tipo del nodo, este es un campo obligatorio y será una cadena de entre las siguientes:
  - *Module*
  - *ClassDef*
  - *Import*
  - *ImportFrom*
  - *Assign*
  - *AnnAssign*
  - *AsyncFunctionDef*
  - *FunctionDef*

- **name**: Representa el nombre del nodo, será una cadena con el nombre del nodo, este es un campo obligatorio.

- **value**: Representará el valor del nodo, será una cadena, un entero o una lista con cadenas, enteros o nodos *PythonNode*. El valor de un nodo dependerá del tipo del mismo:
  - *ImportFrom*: Lista de cadenas de las funcionalidades importadas
  - *Assign*: Valor de la asignación
  - *AnnAssign*: Valor de la asignación
  - *AsyncFunctionDef*: Valor de lo que retorna la función.
  - *FunctionDef*: Valor de lo que retorna la función.

- **args**: Representará los parámetros de una función o la herencia en el caso de clases.
  - *ClassDef*: Lista con representación en formato cadena de clases de las cual hereda.
  - *AsyncFunctionDef*: Lista con parámetros de la función.
  - *FunctionDef*: Lista con parámetros de la función.

- **body**: Representará el cuerpo de una función, una clase o un módulo.
  - *Module*: Lista de nodos que representan el cuerpo del módulo.
  - *ClassDef*: Lista de nodos que representan el cuerpo de la clase.
  - *AsyncFunctionDef*: Lista de nodos que representan el cuerpo de la función.
  - *FunctionDef*: Lista de nodos que representan el cuerpo de la función.

Apéndice II

Glosario de Términos

- **AST**: *Syntax Abstract Tree.* Árbol sintáctico abstracto.
- **CAS**: *Class Average Score.* Puntuación media de las clases.
- **CBO**: *Coupling Between Object classes*. Acoplamiento entre clases.
- **CCD**: *Code Comments Density*. Densidad de comentarios en el código.
- **DIT**: *Depth of Inhetiance Tree.* Profundidad del árbol de herencia.
- **IDE**: *Integrated Development Enviroment*. Entorno de desarrollo integrado.
- **LCOM**: *Lack of Cohesion in Methods*. Carencia de cohesion en los métodos.
- **NOC**: *Number Of Children*. Número de hijos.
- **OOP**: *Object Oriented Programming*. Programación orientada a objetos.
- **UML**: Lenguaje Unificado de Modelado.

Apéndice III

Ejemplos de ejecución

Los ejemplos descritos en las imágenes siguientes han sido generados a partir del código fuente del proyecto de este TFG. El código está alojado en un repositorio de *GitHub [7].*

Ejemplo Diagrama de clases 1

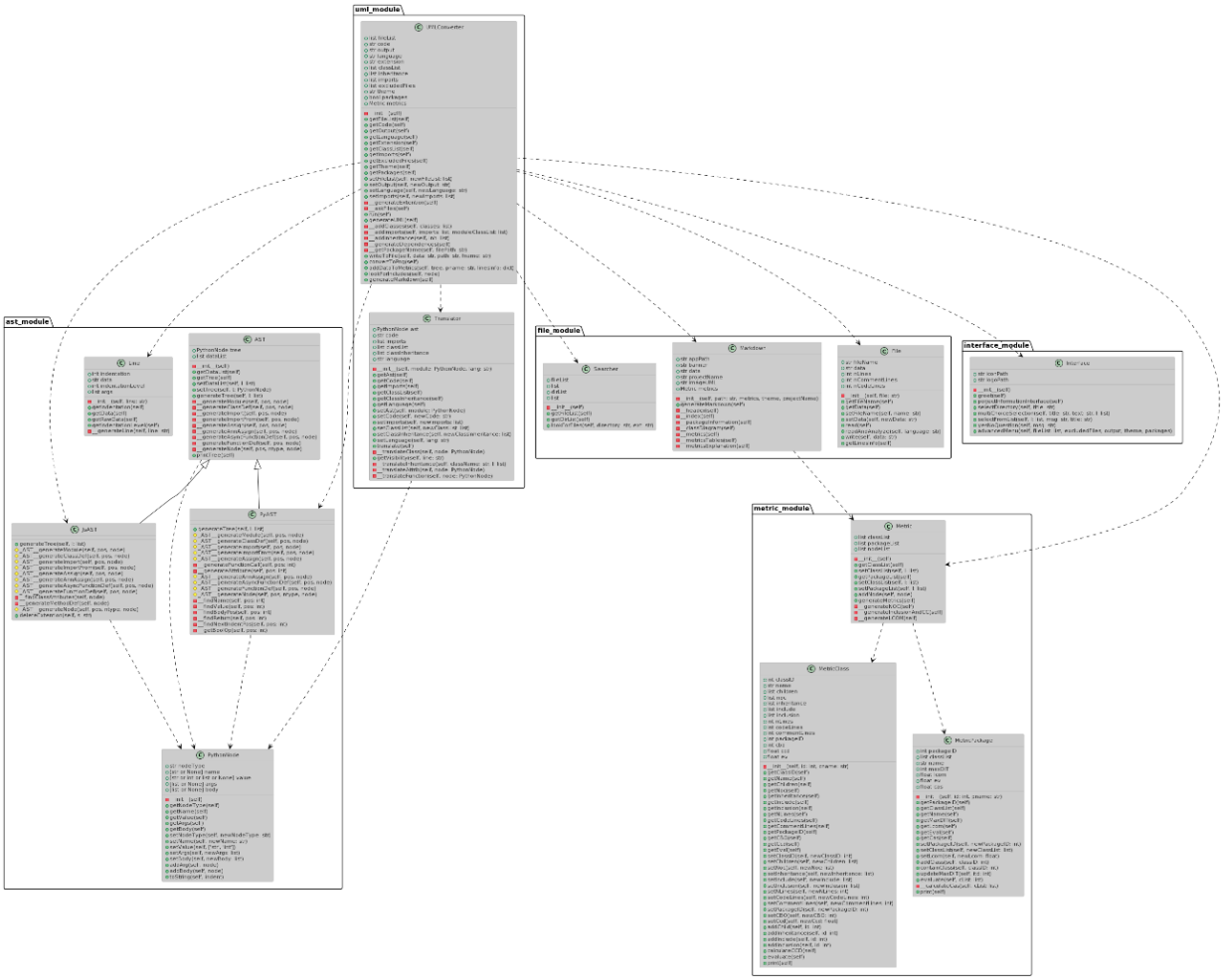

Figura 9: Ejemplo de imagen con diagrama de clases generada sobre el proyecto UMLConverter con agrupamiento en paquetes y con tema "reddress-lightblue".

Ejemplo Diagrama de clases 2

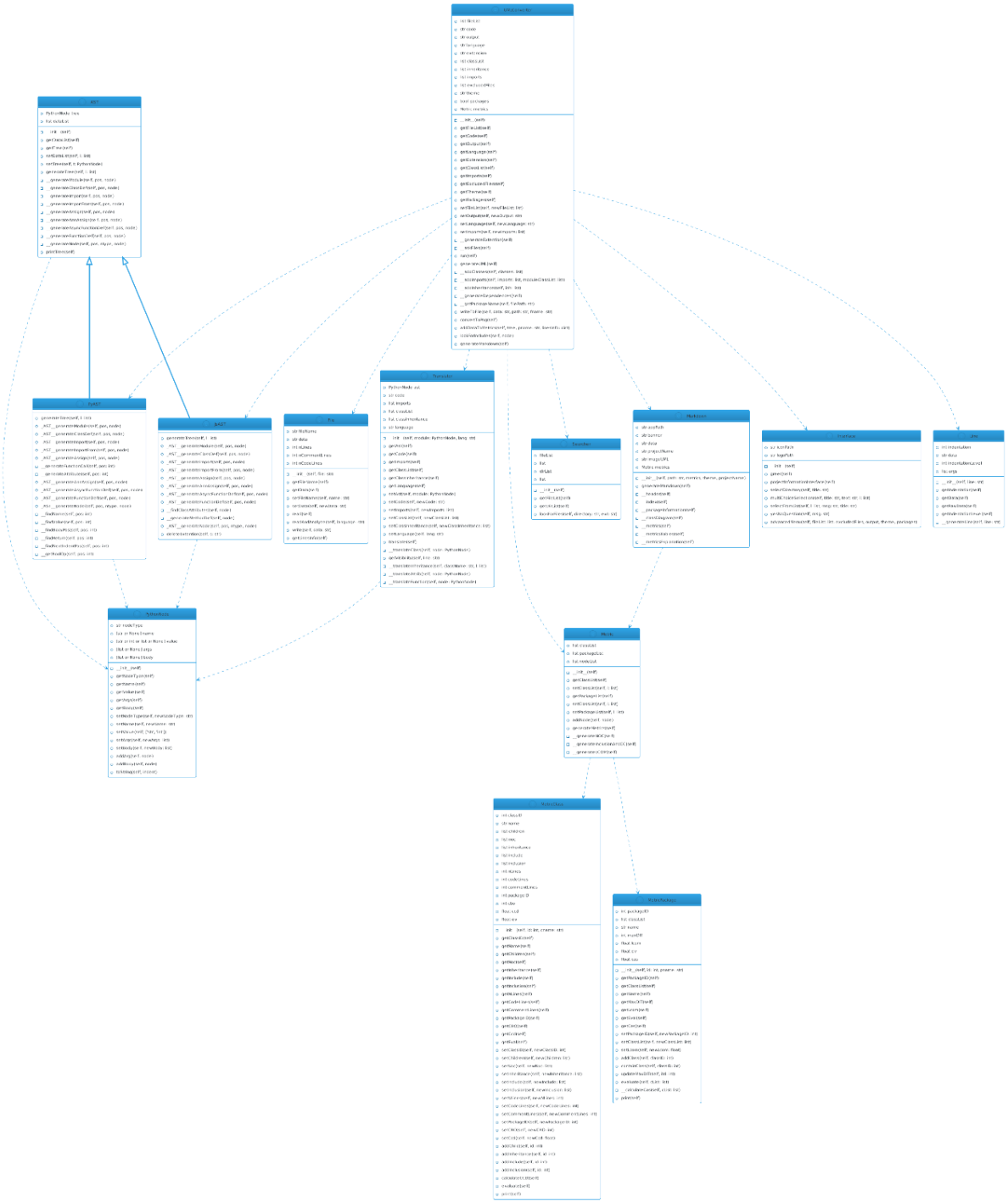

Figura 10: Ejemplo de imagen con diagrama de clases generado sobre el proyecto UMLConverter sin agrupamiento en paquetes y con tema "cerulean-outline".

Apéndice IV

Relación con las materias cursadas

Durante el transcurso de este TFG han sido de gran utilidad los conceptos aprendidos y las competencias adquiridas en numerosas asignaturas que he cursado a lo largo del grado en ingeniería Informática.

- **Cálculo:** Esta asignatura me ha ayudado a la comprensión de forma sencilla y rápida de los cálculos necesarios para el desarrollo de este TFG, por ejemplo, los cálculos necesarios para generar las métricas de las clases y paquetes.

- **Álgebra:** Esta asignatura ha sido de ayuda para aportarme agilidad a la hora de trabajar con las estructuras de datos basadas en vectores. Tener la capacidad de visualizar un vector y moverse por el de forma sencilla mentalmente es necesario para agilizar el trabajo y poder trabajar con ellos de forma abstracta.

- **Algoritmos y Estructuras de Datos:** Esta asignatura me ha aportado facilidad a la hora de crear un buen diseño para la de estructuras de datos.

- **Computabilidad y Algoritmia:** Esta asignatura me ha ayudado al diseño de algoritmos y autómatas en *Python* necesarios para el desarrollo de este TFG. Por otro lado, me ha aportado la comprensión de la importancia de la documentación en un proyecto informático.

- **Inglés técnico:** Esta asignatura me ha permitido documentarme sobre conceptos nuevos o que necesitaban ser ampliados ya que las documentaciones de los lenguajes de programación suelen estar en inglés.

- **Fundamentos de Ingeniería del Software:** Esta asignatura me ha ayudado a comprender las utilidades e importancia de las metodologías de programación y esquematizar el diseño de un proyecto previo al desarrollo del mismo. Personalmente creo que es gracias a esta asignatura por la cual decidí realizar este TFG. He comprendido gracias a ella, la importancia y facilidad que aporta a los programadores el disponer de herramientas como esta a la hora de incorporarse a un nuevo proyecto.

- **Lenguajes y Paradigmas de Programación:** Esta asignatura me ha aportado los conocimientos necesarios para la adaptación a un lenguaje interpretado, como es el caso de *Python*.

- **Procesadores de Lenguajes:** Los conocimientos adquiridos en esta asignatura han sido esenciales para poder desarrollar este TFG ya que toda la información se extrae de un AST, y el tener los conocimientos para trabajar con ellos y facilidad para entenderlos ha sido gracias a esta asignatura.

- **Programación de Aplicaciones Interactivas:** Gracias a esta asignatura he obtenido los conocimientos para para desarrollar una interfaz interactiva para la aplicación.

- **Tratamiento Inteligente de Datos:** Esta asignatura me ha aportado la agilidad para trabajar con la programación en *Python* y posibles usos de este lenguaje para facilitar muchas tareas relacionadas con este trabajo.

Bibliografía

[1] *Python (version 3.10). Accessed: 19th october 2022. Avalaible:* https://www.python.org/downloads/release/python-3100/

[2] *Eclipse (year 2022). Accessed: 20th october 2022. Avalaible:* https://www.eclipse.org

[3] *OMG. Accessed 8th march 2023. Availabe:* https://www.omg.org/spec/UML/

[4] *Umbrello Project (year 2022). Accessed: 16th november 2022. Avalaible:* https://umbrello.kde.org

[5] *GitUML (year 2022). Accessed: 18th november 2022. Avalaible:* https://www.gituml.com

[6] *Sublime Text. Accessed 8th march 2023. Availabe:* https://www.sublimetext.com

[7] *GitHub. Accessed 8th march 2023. Availabe:* https://github.com/AdrianEpi/UMLConverter

[8] *Venv (version 3.3). Accessed: 19th october 2022. Avalaible:* https://docs.python.org/3/library/venv.html

[9] *Pytest (version 7.2.0). Accessed: 26th december 2022. Avalaible:* https://pypi.org/project/pytest/

[10] *Coverage (version 7.2.1). Accessed: 16th november 2022. Avalaible:* https://pypi.org/project/coverage/

[11] *PlantUML (version 0.3.0). Accessed: 18th january 2022. Avalaible:* https://plantuml.com

[12] *Easygui (version 0.98.3). Accessed: 19th december 2022. Avalaible:* https://pypi.org/project/easygui/

[13] *Esprima (version 6.5.0). Accessed: 21st ovember 2022. Avalaible:* https://docs.esprima.org/en/latest/syntax-tree-format.html

[14] *JavaScript. Accessed: 8th march 2022. Avalaible:* https://developer.mozilla.org/es/docs/Web/JavaScript

[15] *AST (version 3.10). Accessed: 27th november 2022. Avalaible:* https://docs.python.org/3.10/library/ast.html#ast.parse

[16] *Mermaid. Accessed: 8th march 2022. Avalaible:* https://mermaid.js.org/intro/

[17] *M. A. Sicilia (2022, Sep. 8). Accessed: 10th February 2023 Internet Archive, Avalaible:* https://archive.org/details/cnx-org-col10583/mode/2up

[18] *Talent. Accessed: 2nd march 2023. Availabe:* https://es.talent.com/salary