



GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA
TELECOMUNICACIÓN

Curso Académico 2016/2017

Trabajo Fin de Grado

ANÁLISIS DE TESTS UNITARIOS DE
PROYECTOS ESCRITOS EN PYTHON

Autor : Alberto Rodríguez Tena

Tutor : Dr. Gregorio Robles

Trabajo Fin de Grado

ANÁLISIS DE TESTS UNITARIOS DE PROYECTOS ESCRITOS EN PYTHON

Autor : Alberto Rodríguez Tena

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Trabajo Fin de Grado se realizó el día de
de 2017, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2017

*Dedicado a
mis padres, mis hermanos y mi novia*

Agradecimientos

Este trabajo pone fin a una de las etapas más importantes y enriquecedoras de mi vida, no solo ha estado marcada por momentos buenos sino también momentos de lucha y crecimiento que hoy veo reconocidos con la entrega de este trabajo y la finalización de esta etapa. Esta etapa además de horas de estudio, esfuerzo y sacrificio me ha dejado muchas personas que han estado apoyándome a lo largo de esta dura etapa y a las cuales no me quiero olvidar de dar las gracias en este trabajo, los primero y mas importantes a mis padres por estar apoyándome siempre en los momentos difíciles y que sin ellos esto no hubiera sido posible dado que me han ayudado a ser quién soy ahora. También y como no mencionar a mi tutor por el esfuerzo, dedicación y constancia que ha tenido conmigo en las realización de este trabajo y que me ha ayudado a crecer tanto a nivel personal como profesional ayudándome a poner fin a esta etapa que siempre recordaré con cariño.

Resumen

Este trabajo fin de grado analiza proyectos Python de GitHub de acceso público para saber cual es el porcentaje de funciones que contiene el proyecto y que se están comprobando con tests unitarios además, de saber cuáles son las funciones que se han comprobado y de que manera. Con el fin de saber que partes de nuestro código están siendo comprobadas para asegurarse de su correcto funcionamiento.

En el documento se explicará el entorno de los test unitarios en Python y en que medida estos tests son utilizados debido a sus beneficios a la hora de realizarlos, por la importancia que tiene la comprobación del código y para saber si la solución escogida es la correcta.

Este trabajo se ha elaborado mediante una aplicación web escrita en el lenguaje de programación Python utilizando el *framework* web de Python, Django además de otras librerías y otras tecnologías como pueden ser Ctags, CSS3 y HTML5. Los datos que se obtendrán del análisis serán almacenados en una base de datos SQLite.

Summary

This project will analyze public access Python projects from GitHub to know the percentage of functions contained in each project and that are being tested in unit tests as well as which of the functions are being tested and how, in order to know which parts of the code are being checked to ensure that they work correctly.

In the project explained below it is presented the environment of the unit tests in Python and how much those tests are used, because of its benefits when developing them to verify the code in order to know if the chosen solution is correct.

This project has been developed using a web application written in Python programming language using the Python web framework, Django in addition to other libraries and other technologies such as Ctags, CSS3 and HTML5. The data obtained from the analysis will be stored in a SQLite database.

Índice general

1. Introducción	1
1.1. Contexto	1
1.2. Motivación	2
1.3. Estructura de la memoria	2
2. Objetivos	5
2.1. Objetivo general	5
2.2. Objetivos específicos	5
2.3. Planificación temporal	6
3. Estado del arte	9
3.1. Python	9
3.2. Django	10
3.3. SQLite	11
3.4. Git	11
3.5. GitHub	12
3.6. Ctags	12
3.7. HTML5	13
3.8. CSS3	13
3.9. JavaScript	14
3.10. BootStrap	14
4. Diseño e implementación	17
4.1. Arquitectura general	17
4.2. Back-end	18

4.2.1. Análisis de proyectos	18
4.2.2. Modelo de datos	21
4.2.3. Cálculo de porcentajes	23
4.3. Front-end	23
5. Resultados	27
6. Conclusiones	31
6.1. Consecución de objetivos	31
6.2. Aplicación de lo aprendido	32
6.3. Lecciones aprendidas	32
6.4. Trabajos futuros	33
6.5. Valoración personal	33
A. Manual de usuario	35
Bibliografía	37

Índice de figuras

3.1. Logo de Python	10
3.2. Logo de Django	10
3.3. Logo de SQLite	11
3.4. Logo de Git	12
3.5. Logo de GitHub	12
3.6. Logo de HTML5	13
3.7. Logo de CSS3	13
3.8. Logo de JavaScript	14
3.9. Logo de BootStrap	15
4.1. Estructura de la aplicación web	17
4.2. Casos contemplados y no contemplados en el trabajo.	20
4.3. código para generar las tablas	22
4.4. Página principal	24
4.5. Popup cuando se quiere analizar un proyecto ya existente	24
4.6. Ficheros de un proyecto	25
4.7. Funciones y test de un fichero	25
5.1. Porcentajes de los repositorios analizados	28
5.2. Test unitario ejecutando varias funciones	29
5.3. Funciones del repositorio <i>basic – math – functions – unittest</i>	29

Capítulo 1

Introducción

En este primer capítulo se va a tratar el contexto en el que se ha llevado a cabo este trabajo. A continuación se expondrán las distintas motivaciones que han llevado a realizarlo y en último lugar se abordará que estructura va a seguir esta memoria.

1.1. Contexto

En el mundo de la programación, como regla general, se solucionan problemas de diversos tipos. Por ello surge la siguiente cuestión, de qué manera realmente se prueba que la solución escogida es la correcta. Con el motivo de dar respuesta a esta cuestión surge la realización de este trabajo, el cual dirá si se está testeado o no el software. En caso afirmativo nos dirá de qué manera se está probando dicho software.

Para resolver la cuestión anterior en este trabajo se van a analizar diferentes proyectos Python de GitHub. Para saber qué porcentaje de funciones contenidas en dichos proyectos están siendo testeadas, con la finalidad de saber qué parte del software ha sido comprobado. El motivo por el cual es necesario realizar este análisis es que un proyecto siempre tendrá partes más críticas. En estas partes es aconsejable la realización de la comprobación del código implementado. Por ejemplo, puede ser una simple suma pero si es un programa para hacer transferencias bancarias y nuestra solución para dicho problema suma o resta mal la consecuencia puede ser bastante mayor.

Por esta razón se ha diseñado una metodología de diseño de software llamada TDD (Test-Driven Development). En la que se realizan primero los test (la solución que se quiere obtener

del software ante unos datos) y luego se implementa el software para poder pasar dichos tests. Esta metodología se está implantando en muchas empresas de desarrollo de software para no vender un código que esté mal implementado y que pueda generar errores en un futuro.

1.2. Motivación

La realización de este trabajo ha estado motivada por conocer si realmente estas metodologías se están implementando en la vida real. Para ver cuanta parte del software que se utiliza ha sido comprobado, es decir, que funciona realmente como se quiere que funcione.

Además al tratarse de una aplicación web incluye otros aspectos como son el desarrollo web, de software y técnicas de BI (business intelligence) para el análisis y estudio de los datos obtenidos.

1.3. Estructura de la memoria

En esta sección se describirá la estructura de la memoria:

- En el capítulo 1 se realizará una explicación del contexto y la motivación para llevar a cabo el trabajo.
- En el capítulo 2 se determinan los objetivos tanto a nivel general como específico del trabajo.
- En el capítulo 3 se hará una explicación de las diferentes tecnologías utilizadas.
- En el capítulo 4 se desarrolla la arquitectura general del trabajo explicando sus diferentes partes.
- En el capítulo 5 se detallarán los resultados obtenidos.
- En el capítulo 6 se explica que objetivos de los explicados se han conseguido llevar a cabo.
- En el apéndice A se muestra una guía de instalación y ejecución para el usuario.

- En la bibliografía se listan las fuentes de información utilizadas para la elaboración de esta memoria.

Capítulo 2

Objetivos

En este capítulo se van a tratar los objetivos por los cuales se ha realizado este trabajo de fin de grado.

Para poder comprender este trabajo y antes de pasar a explicar tanto los objetivos generales como específicos es necesario hacer una explicación de qué es un test unitario en Python y cuáles son sus utilidades. Estos tests se definen como la forma de comprobar el correcto funcionamiento de una unidad de código en diseño funcional se corresponde con una función. Para elaborar un test unitario en Python es necesario crear una clase que herede de `unittest.TestCase` y dentro de esta clase elaborar métodos que empiecen por la palabra `test`.

2.1. Objetivo general

El objetivo general de este trabajo es ver qué porcentaje de funciones de un proyecto Python subido en GitHub están testeadas. En el cual además de ver el porcentaje se podrá ver que función ha sido testada y con que test, así como ver que comparación se ha realizado en dicho test para comprobar la función. Esta comparación se realiza con los distintos `self.assert` que ofrece la librería de Python `unittest`.

2.2. Objetivos específicos

El objetivo general se divide en una serie de objetivos específicos que son los siguientes:

- Elaboración de un proyecto en Django que es dónde se encuentra la aplicación web.

- Familiarizarse con la estructura que sigue un programa en Python.
- Analizar cada código Python para extraer sus características más importantes. Para ello se ha utilizado el programa ctags el cual generará un fichero con las características del código.
- Crear el modelo de base de datos y almacenar las características obtenidas gracias al parseo del fichero que genera el ctags.
- Utilización de métodos de BI para el análisis de los datos almacenados en la base de datos.
- Una vez se han obtenido los resultados se muestran en la página web. Para la cual se ha elaborado un CSS y un JavaScript para mejorar la estética.
- Analizar el mayor número de proyectos posibles para obtener el mayor número de datos y así poder sacar mejores conclusiones.

2.3. Planificación temporal

La planificación seguida para la realización de este trabajo es la siguiente:

- Reunión con el tutor para ver cual va a ser el tema a tratar así como los objetivos y que finalidad se busca con este trabajo.
- Una vez acordado lo anterior, búsqueda de la información necesaria para la familiarización con el entorno.
- Realización de las primeras pruebas y elaboración de una aplicación de prueba.
- Una vez realizada la aplicación de prueba, reunión con el tutor para elegir la arquitectura de la aplicación web.
- Realización del código que implementa el análisis de la aplicación, realizando diversas pruebas con diferentes proyectos Python de GitHub.
- Elaboración del modelo de datos para el posterior almacenamiento de los datos obtenidos a partir del análisis.

- Utilización de técnicas de BI para mostrar los resultados obtenidos en la página web. Además con el fin de mejorar la estética de la página web se elaboró un CSS y un JavaScript.

Capítulo 3

Estado del arte

En este capítulo se va a describir las tecnologías que han sido utilizadas para la realización de este trabajo. Estos conceptos son importantes para tener una mejor comprensión del trabajo.

3.1. Python

Python [9] es un lenguaje de programación de propósito general, orientado a objetos, programación imperativa y en menor medida programación funcional. Debido a las características citadas anteriormente se puede decir que Python es un lenguaje de uso sencillo y de fácil aprendizaje. Por ello, fue la opción mas idónea de entre los lenguajes de programación para llevar a cabo este trabajo.

Fue creado por Guido Van Rossum, con el el objetivo de cubrir la necesidad de un lenguaje orientado a objetos de sencillo uso que se podrá utilizar para tratar diversas tareas dentro de la programación. Dado que estas tareas habitualmente se hacían en Unix usando C.

Python es un lenguaje de scripting, independiente de plataforma y como ya se ha comentado orientado a objetos. Esto le permite a este lenguaje estar preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso páginas web. Es un lenguaje interpretado esto quiere decir que no se necesita compilar el código fuente para poder ejecutarlo, esto ofrece grandes ventajas como la rapidez de desarrollo.

Python dispone de dos versiones que son usadas en la actualidad la 2.x y la 3.x esta última incluye una serie de cambios que hacen necesario tener que reescribir el código de versiones anteriores. En este proyecto se ha utilizado la versión 2.7.6 de Python.

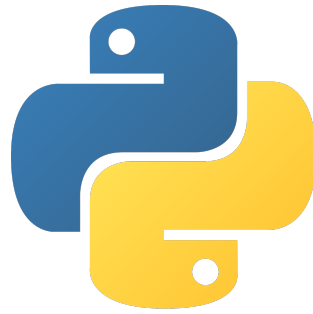


Figura 3.1: Logo de Python

3.2. Django

Django [4] es un framework (conjunto de componentes que te ayudan a desarrollar sitios web más fácil y rápidamente) para aplicaciones web. Gratuito y open source escrito en Python.

Fue desarrollado en principio para gestionar varias páginas orientadas a noticias de World Company de Lawrence, Kansas y fue liberada al público bajo una licencia de BSD (Distribución de Software Berkeley).

El principal objetivo que tiene Django es facilitar la creación de sitios web complejos. Django pone hincapié en reusar la conectividad y extensibilidad de componentes de desarrollo rápido y el principio DRY (No te repitas). Python es utilizado en todas las partes de framework, incluso en configuraciones, archivos y en los modelos de datos.

Django requiere de una versión de Python 2.5 o superiores en este proyecto se ha utilizado la versión de Python 2.7.6. No se necesitaran por tanto otras bibliotecas de Python para obtener una funcionalidad básica. Para la realización de este proyecto se ha utilizado la versión de Django 1.9.5.



Figura 3.2: Logo de Django

3.3. SQLite

SQLite [10] es una biblioteca escrita en C que implementa un motor de base de datos SQL de dominio público, de alta confiabilidad, incorporado y con todas las funciones. Es compatible con ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad).

Es el motor de base de datos mas utilizado en el mundo. A diferencia de los sistemas de gestión de bases de datos cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca de SQLite se enlaza con el programa pasando a ser parte integral del mismo.

En su versión 3, que es la que se ha utilizado para este proyecto SQLite permite bases de datos de hasta 2 Terabytes de tamaño y también permite la inclusión de campos de tipo BLOB (Objetos binarios grandes).



Figura 3.3: Logo de SQLite

3.4. Git

Git [5] es un sistema de control de versión distribuido libre y de código abierto diseñado para manejar de todo, desde proyectos pequeños a muy grandes con rapidez y eficiencia. Fue diseñado por Linus Torvalds pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

Debido a sus características Git es de uso muy sencillo y tiene una huella muy pequeña y un rápido rendimiento. Supera las herramientas de SCM como subversion, CVS, Perforce y ClearCase con funciones como ramificación local barata, áreas de puesta en escena convenientes y múltiples flujos de trabajo.



Figura 3.4: Logo de Git

3.5. GitHub

GitHub [6] es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Utiliza el framework *Ruby on Rails* por GitHub.

El código se almacena de forma pública, aunque tiene la posibilidad de almacenarlo de forma privada. Fue desarrollada por Tom Preston-Werner. GitHub funciona de la siguiente manera aloja un repositorio de código y brinda las herramientas más útiles para el trabajo en equipo, de estas herramientas se puede destacar, wikis, sistemas de seguimiento de problemas, herramientas de revisión de código y visor de ramas.



Figura 3.5: Logo de GitHub

3.6. Ctags

Ctags [3] es una herramienta de programación que genera un archivo de índice de nombres encontrados en archivos de origen y encabezado de varios lenguajes de programación. Estas etiquetas o índices permiten que las definiciones sean localizadas de manera fácil y rápida por un editor de texto u otra herramienta.

El Ctags original fue introducido en BSD Unix y fue escrito por Ken Arnold con apoyo de Fortran Jim Kleckner y de Bill Joy.

3.7. HTML5

HTML5 [7] es el lenguaje de marcado estándar para crear páginas Web. HTML significa Hyper Text Markup Language.

HTML5 es la quinta versión de HTML, con nuevos elementos, atributos y comportamientos. Contiene un conjunto más amplio de tecnologías que permite a los sitios Web y a las aplicaciones ser más diversas y de gran alcance.



Figura 3.6: Logo de HTML5

3.8. CSS3

CSS3 [2] es la última versión de CSS, es un lenguaje que define el estilo o la apariencia de las páginas web, escritas en HTML o documentos XML.

Fue creada para separar el contenido de la forma y esto permite a los diseñadores tener un control mayor de la apariencia de la página.



Figura 3.7: Logo de CSS3

3.9. JavaScript

JavaScript [8] es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

JavaScript es un lenguaje que se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web lo que permite mejoras en la interfaz de usuario y páginas web. Existe además una forma de JavaScript del lado del servidor. También es significativo su uso en aplicaciones externas a la web.

Fue diseñado con una sintaxis muy similar a C, aunque tiene también convenciones y nombres de Java.



Figura 3.8: Logo de JavaScript

3.10. BootStrap

BootStrap [1] es un framework de código abierto para diseño de sitios y aplicaciones web, contiene elementos de diseños basados en HTML y CSS, así como extensiones de JavaScript opcionales.

Es conocido como el proyecto más popular de GitHub, fue desarrollado por Mark Otto y Jacob Thornton de Twitter como un marco de trabajo (framework). Está mantenido por Twitter. Es compatible con la mayoría de los navegadores web.



Figura 3.9: Logo de BootStrap

Capítulo 4

Diseño e implementación

En este capítulo se mostrará la arquitectura del trabajo, así como su forma de implementarlo y la explicación de las partes más relevantes de la aplicación, el front-end y el back-end.

4.1. Arquitectura general

Este trabajo consiste en una aplicación web Django la cual mostrará los resultados del análisis producidos en la parte del back-end. Este resultado será el porcentaje de funciones que se comprueban con tests unitarios dentro de un proyecto Python de GitHub. Teniendo que heredar estos tests de la clase `unittest.TestCase`. A continuación en la figura 4.1 se puede ver un breve esquema de la estructura del proyecto.

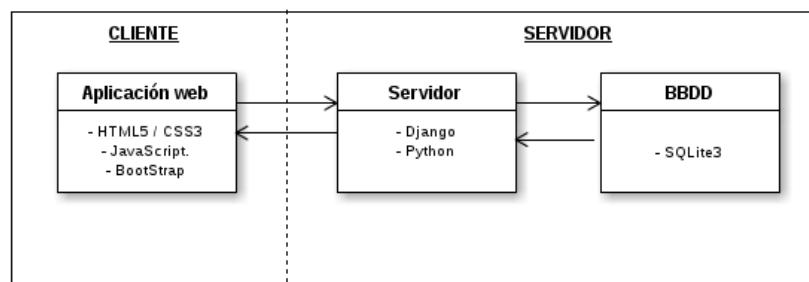


Figura 4.1: Estructura de la aplicación web

Este análisis se realizará en la parte del back-end, la cual ha sido desarrollada en Python mediante el framework web de Python, Django. Esta parte además de realizar el análisis se encarga de acceder a la base de datos y almacenar los datos obtenidos del análisis. Una vez

almacenados los datos, el servidor devolverá estos datos almacenados que alimentarán a la parte del front-end.

La parte del front-end mostrará los resultados de forma sencilla en la página web para poder ser comprendidos por todos los usuarios. Haciendo que estos resultados sean navegables lo cual permite acceder a mas información. Además permite introducir la URL del proyecto GitHub que se quiere analizar.

4.2. Back-end

En este apartado se explicarán las características de la parte del Back-end de la aplicación. La elaboración de esta parte tiene lugar en el servidor Django desde la cual se sirven los contenidos a la página web. Este servidor tiene disponibles tres recursos que son `/`, `/repo/(id)`, `/fich/(id)` a continuación se mostrará que se realiza en cada uno de estos recursos.

- `/`: en este recurso se puede enviar en el cuerpo de la petición HTTP la URL del proyecto que se quiere analizar. Una vez este proyecto haya sido analizado se obtendrán los resultados de dicho análisis, el cual será el porcentaje de funciones que han sido comprobadas con test unitarios (que heredan de la clase de Python `unittest.TestCase`).
- `/repo/(id)`: accediendo a este recurso se devolverán los ficheros que contiene un proyecto con sus porcentajes correspondientes. Haciendo referencia al número de funciones que posee dicho fichero y que se han comprobado con tests unitarios del proyecto.
- `/fich/(id)`: al acceder a este recurso se devolverán las funciones que posee un fichero, así como en que tests unitarios han sido comprobadas en el caso de que así sea.

4.2.1. Análisis de proyectos

En este apartado se explicará como se realiza el análisis para cada uno de los proyectos.

Proceso e implementación

Este proceso de análisis comienza desde que se envía la URL de un proyecto en la cabecera HTTP desde el recurso `/` hasta que se relacionan los tests con la función que están testando.

Cuando se ha recibido la URL del proyecto se procederá a clonar dicho proyecto en local utilizando la herramienta *git clone*. Esta herramienta generará una carpeta con el contenido del proyecto y se procederá a guardar este nombre de proyecto con su URL en la base de datos. Una vez se tiene una copia del proyecto se va a realizar una búsqueda mediante el comando *find* de Unix para encontrar todos los ficheros que terminen en *.py*, es decir, todos los ficheros PYTHON que contiene el proyecto. Los cuales se almacenarán en la base de datos.

Cuando ya se tienen los nombres de todos los ficheros Python del proyecto se realizará la operación de *ctags* para cada uno de ellos. Esta operación lo que hace es generar un fichero con la información más relevante del fichero a la entrada. Analizando este fichero que nos genera la operación *ctags* a través del comando *grep* de Unix se obtendrán por un lado los nombres de las funciones, los cuales, se guardarán en la base de datos y por otro lado el nombre de los tests que contiene el fichero de nuestro proyecto.

Una vez se tiene el nombre de los test se realizará una búsqueda dentro del fichero que contiene el test para saber que función esta testeando. Para ello, se leerá el fichero con el objetivo de encontrar el *self.assert* correspondiente al test, del cual, se obtendrán sus argumentos para saber que función está testeando. Debido a que son tests unitarios solo pueden comprobar una única función. Una vez conseguida la relación entre el test y la función testada se guarda el nombre del test en la base de datos con su correspondiente relación finalizando el proceso de análisis.

Otras consideraciones

Para el funcionamiento de la aplicación es necesario tener en cuenta ciertas limitaciones, debido al amplio abanico de posibilidades que se encuentran a la hora de hacer tests unitarios.

- Por un lado, en el cuadro 4.2 se mostrarán aquellos casos en los que la aplicación funciona y cuales no. Entendiendo que en los casos que no se contemplan no saltarán errores sino que no encontrará la relación entre la función y el test y continuará con el análisis.

CASOS CONTEMPLADOS	CASOS NO CONTEMPLADOS
<ul style="list-style-type: none"> - Solo se podrán usar test dentro de una clase que herede de la clase <code>Unittest.TestCode</code>. - La llamada al <code>self-assert</code> tiene que estar en una única línea, sin salto de línea entre medias. - Las clases se tienen que importar de una en una es decir en diferentes líneas. - <i>Funciones</i>¹ que se encuentran dentro del mismo fichero en el que está el test. - <i>Funciones</i>² que se encuentran en clases fuera del fichero donde esta el test, pero como requisito estas clases tienen que estar dentro de la misma carpeta donde esta el fichero que contiene al test. 	<ul style="list-style-type: none"> - Cualquier test no heredado de <code>Unit-test.TestCode</code>. - No entiende el <code>self.assertRaises</code>. - Comparaciones con funciones predefinidas en Python. - No se puede comprobar más de una función dentro de un test debido a que son test Unitarios que solo comprueban una función.

Figura 4.2: Casos contemplados y no contemplados por el trabajo.

1. Se puede realizar un test de una función que se encuentre en el mismo fichero ya sea dentro o fuera de una clase. Se puede llamar desde el `self.assert` de forma directa como podría ser por ejemplo, `self.assertEqual(funcion(), expected)` o igualando una variable al resultado de la función como podría ser `x = funcion()`. Una vez está igualado se llamará a la `x` dentro del `self.assert` quedando de esta forma, `self.assertEqual(x, expected)`.

2. Sí la función que se quiere llamar se encuentra dentro de una clase que no esta en el mismo fichero que esta el test hay dos opciones posibles que contempla este trabajo.

- Por un lado crear el objeto dentro del `self.assert` y llamar a la función dentro también del mismo o también se puede igualando una variable a dicha función y llamarla dentro del `self.assert` como en el caso anterior. Como se puede observar en el siguiente ejemplo:

```
self.assertEqual(Clase().funcion(), expected)
o bien
objeto = Clase().funcion()
self.assertEqual(objeto, expected)
```

- Y por otro lado crear un objeto fuera del self.assert y una vez se tiene este objeto, desde este se hace la llamada a la función pudiéndolos meter dentro del self.assert o igualando esto a una variable y llamarla dentro del self.assert, como se explica en el siguiente ejemplo:

```
objeto=Clase()
self.assertEqual(objeto.funcion(), expected)
o bien
x=objeto.funcion()
self.assertEqual(x, expected)
```

- Por otro lado, se encuentra otra limitación en el momento de usar ctags debido a que cuando el fichero contiene un string comprendido entre ''' lo considera como un comentario entonces el fichero que genera es erróneo.

4.2.2. Modelo de datos

En este apartado se va a mostrar que tablas y que relaciones hay entre las tablas de la base de datos. En la figura 4.3 se representa el código para generar dichas tablas y sus relaciones.

```

class Repositorios (models.Model):
    nombre = models.CharField(max_length = 200)
    url = models.CharField(max_length = 200)
class Ficheros (models.Model):
    nombre = models.CharField(max_length = 200)
    repositorio = models.ForeignKey(Repositorios)
    porcentaje = models.CharField(max_length = 20)
class Funciones (models.Model):
    nombre_funcion = models.CharField(max_length = 200)
    fichero = models.ForeignKey(Ficheros)
    testeada = models.CharField(max_length = 2,default='N0')
    linea = models.CharField(max_length = 6)
class Funciones_test (models.Model):
    nombre_test = models.CharField(max_length = 200)
    funcion_testeada = models.CharField(max_length = 200)
    fichero_testeado = models.ForeignKey(Ficheros, related_name='testeado')
    fichero_propietario = models.ForeignKey(Ficheros, related_name='propietario')
    comparacion = models.CharField(max_length = 200)
    pertenece_funcion = models.CharField(max_length = 2,default='N0')
    linea = models.CharField(max_length = 6)

```

Figura 4.3: código para generar las tablas

Como se puede observar en la figura 4.3 se tienen cuatro tablas diferentes que son:

- Repositorios: en esta tabla se van almacenar los nombres de los proyectos analizados con su correspondiente URL que vendrá en el cuerpo de la petición HTTP desde el recurso /.
- Ficheros: en esta tabla se almacenan los nombres de todos los ficheros Python correspondientes a los proyectos analizados. Para ello hay una ForeignKey que relaciona esta tabla con la de Repositorios y además el campo porcentaje dice que funciones de este fichero se han comprobado con tests unitarios.
- Funciones: esta tabla contiene el nombre de las funciones correspondientes a cada uno de los ficheros. Para ello hay una ForeignKey la cual relaciona esta tabla con la tabla de Ficheros. Además almacenará en el campo testeada si esta o no testeada esta función y en el campo línea se guarda el número de línea en la que aparece en el fichero.
- Funciones test: donde se almacenan los nombres de los tests que aparecen en cada uno de los ficheros. En el campo fichero propietario se va a almacenar el nombre del fichero que contiene dicho test. En el campo fichero testeado se va a almacenar el fichero que contiene la función que se está comprobando. En el campo función testeada se va a guardar la función que se está comprobando en el test. En el campo comparación se va a almacenar que

`self.assert` está realizando este test con sus argumentos. En el campo pertenece función se pondrá un si o no dependiendo de si este test está comprobando una función que se tenga almacenada en la tabla de Funciones.

4.2.3. Calculo de porcentajes

Una vez explicado en el punto anterior de que modo se almacenan los datos y en que tablas, se procede a explicar de que manera se calcula el porcentaje para cada uno de los elementos.

- **Porcentaje proyecto:** Para obtener el porcentaje de funciones que se comprueban en tests unitarios de un proyecto se va a realizar una query. Esta query devolverá el número de funciones que se han comprobado con tests unitarios. Este número se va dividir entre el número de funciones totales que contiene el proyecto.
- **Porcentaje fichero:** Para obtener el porcentaje de funciones que se comprueban en tests unitarios de un fichero se va a realizar una query. Esta query devolverá todas las funciones de dicho fichero que se estén comprobando con tests unitarios. Este número se dividirá entre el número de funciones totales que contiene el fichero.

4.3. Front-end

Es la parte que el cliente ve de la aplicación, es decir, la página web. Esta, está elaborada con HTML5, CSS3 y JavaScript para darle un mejor diseño y una fácil utilización. En la figura 4.4, se puede ver la página principal de la aplicación en la cual se puede observar un input que será el lugar donde se introduzca la URL del proyecto que se quiera analizar. Una vez esté puesta la URL en el input se pulsará el botón Analizar para hacer el análisis del proyecto escogido.

En la parte inferior se pueden ver los proyectos que han sido analizados con sus correspondientes porcentajes. Cuando se quiere analizar un proyecto que ya ha sido analizado saltará un popup para preguntar si se quiere sobrescribir el análisis anterior como se puede ver en la figura 4.5. Por lo que habrá dos opciones posibles, una darle al botón cancelar por lo que no volverá a analizar este proyecto o bien darle al botón aceptar lo que conllevará la eliminación del análisis anterior y procederá a analizarlo de nuevo.



Introduza la url del proyecto:

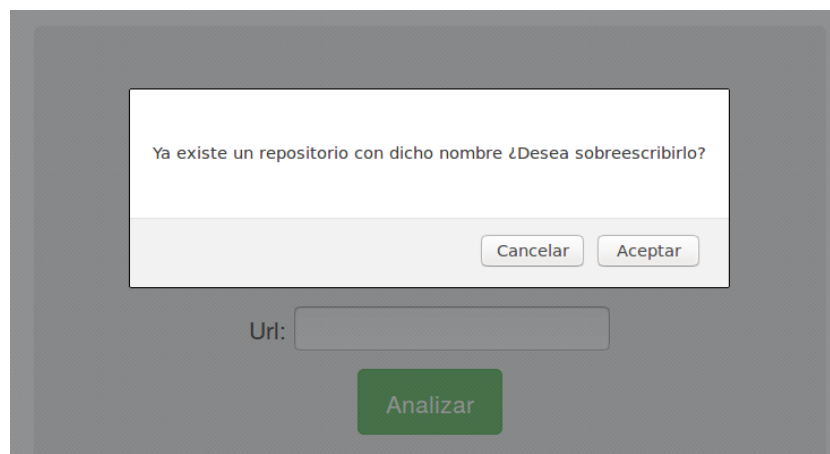
Url:

Analizar

Test_python
50.0%

Test_python_2
18.18%

Figura 4.4: Página principal



Ya existe un repositorio con dicho nombre ¿Desea sobreescribirlo?

Cancelar Aceptar

Url:

Analizar

Figura 4.5: Popup cuando se quiere analizar un proyecto ya existente

Una vez se ha analizado el proyecto escogido después de esperar unos segundos, aparecerá en la parte inferior izquierda el nombre del proyecto. Este será un link que redirigirá a la página en la cual se muestran todos los ficheros Python del proyecto que contengan funciones, con sus correspondientes porcentajes. Como se puede observar en la figura 4.6

Ficheros en Test_python_2



Figura 4.6: Ficheros de un proyecto

Cada fichero será un link a la página donde se mostrarán las funciones que contiene y si están testeadas o no. En caso afirmativo aparecerá un visto al lado de la función. Si se coloca el puntero encima se mostrará que test es el que está testeando dicha función. En el caso contrario aparecerá una cruz roja.

En algunos casos el fichero contiene tests que testean varias veces la misma función con diferentes pruebas, o realizan tests sobre variables. Estos tests aparecerán en la parte inferior bajo el título de *Otros test*. Todo esto se puede observar en la siguiente figura 4.7. Todas las funciones y tests son enlaces a ellas mismas en el proyecto en GitHub.



Figura 4.7: Funciones y test de un fichero

Capítulo 5

Resultados

En este capítulo se explicarán los distintos resultados que se han obtenido del análisis de diferentes proyectos Python de GitHub realizados por diferentes personas y de acceso público.

Tras este análisis, se puede destacar que los desarrolladores utilizan tests unitarios. Pero dentro de estos tests ejecutan varias funciones y como son tests unitarios estos tests solo comprueban una única función. Por lo tanto todas esas funciones que se ejecutan antes de nuestra comprobación (`self.assert`) no se consideran funciones testeadas. Solo se considera que la función esta testada si se encuentra dentro de la comprobación.

Haciendo un estudio en ciertos proyectos se puede ver que los desarrolladores implementan tests unitarios pero no ejecutan una única función que sería lo correspondiente con un verdadero test unitario sino que implementan tests unitarios que ejecutan varias funciones. Debido a que el testeo por tests que solo ejecuten una función es más pesado, porque una función es igual a un test. Pero es una mejor manera de comprobar el código, porque si falla el test se va a saber que función no ha funcionado como se esperaba debido a que solo se ejecuta una única función en el test. Pero si se implementan los tests por el otro método en el caso de que falle no se va a saber donde está realmente el error debido a que está ejecutando varias funciones. Esto implicará hacer una división de este test en tantos tests unitarios como funciones ejecutadas contenga dicho test debido a que estos tests unitarios con el fin de detectar el error solo pueden ejecutar una única función para averiguar cual de las funciones esta generando el error.

Un ejemplo puede ser un proyecto que se llevó a cabo en la Universidad Rey Juan Carlos, llamado `ISI_Project` que fue desarrollado mediante la metodología TDD en el que se observará lo explicado anteriormente. Solo un 20 % de las funciones de este proyecto han sido comprobadas

con tests unitarios. Porque si un proyecto tiene muchas funciones, será más cómodo elaborar tests que ejecuten varias funciones y solo en el caso de que haya fallo en el test se recurrirá a los tests unitarios que ejecutan una única función para detectar el error.

Con relación a lo explicado anteriormente se puede concluir que los tests unitarios que solo ejecutan una única función son útiles para detectar errores pero no como metodología para comprobar un proyecto entero. Debido a que serían numerosos tests y al ser un número tan elevado se acabarían cometiendo errores en la elaboración de los tests. Por tanto se puede considerar una buena herramienta de detección de errores para aquellos casos que los test más generales fallen.

En la siguiente figura 5.1 se puede ver el porcentaje obtenido al analizar cuatro proyectos Python de GitHub los cuales corroboran lo explicado anteriormente.

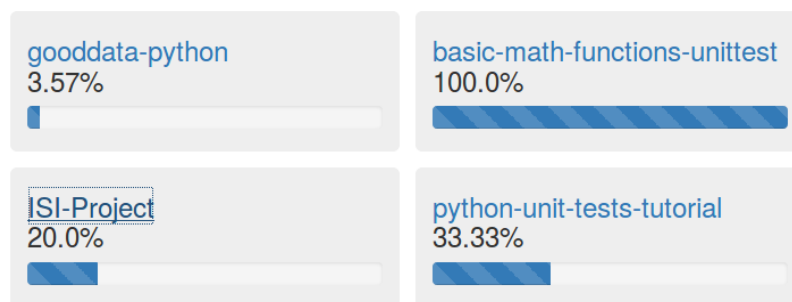


Figura 5.1: Porcentajes de los repositorios analizados

Como se puede observar en los resultados obtenidos se pueden distinguir dos casos. Uno de ellos engloba los porcentajes de 3.57 %, 20 % y 33.33 % los cuales coinciden con lo explicado anteriormente. En la figura 5.2 se puede observar como un único test unitario esta ejecutando varias funciones. Este test corresponde con el proyecto ISI-Proyect. Y en el otro caso si se mira lo que hay dentro del proyecto *basic-math-functions-unittest* como se observa figura 5.3 se verá que esta formado por unas pocas funciones de operaciones simples. Por este motivo se realizan tests unitarios sobre cada una de ellas lo que conlleva a obtener como resultado un 100 %.

```
def test_todas_pos_validas(self):
    t = Tablero()
    ficha1 = ArrayFichas().sacar_ficha(0) #Todo aldea
    t.insertar(ficha1, 2, 2)
    ficha2 = ArrayFichas().sacar_ficha(22) #Solo un lado con aldea
    expected = [[3, 2]]
    self.assertEqual(expected, t.todas_pos_validas(ficha2))
```

Figura 5.2: Test unitario ejecutando varias funciones

Funciones en operators.py

addition	✓	divide	✓	multiply	✓	subtract	✓
----------	---	--------	---	----------	---	----------	---

Figura 5.3: Funciones del repositorio *basic – math – functions – unittest*

Capítulo 6

Conclusiones

En este capítulo se explicará la consecución de objetivos teniendo en cuenta los objetivos explicados anteriormente, qué lecciones se han aprendido una vez realizado este trabajo seguida de una valoración personal.

6.1. Consecución de objetivos

En este apartado se abordarán que objetivos de los explicados en el capítulo 2 han sido realizados con éxito en este trabajo, concluyendo que todos estos objetivos se han alcanzado de manera satisfactoria.

- Se ha conseguido realizar una aplicación en Django, que es capaz de interactuar con una base de datos y volcar los resultados en una página web.
- Se consiguió entender que estructura sigue un programa en Python para poder sacar las características mas relevantes de su código con ayuda del programa ctags.
- Se ha conseguido realizar técnicas de BI para mostrar los datos almacenados que previamente se habían requerido.

Por último es importante mencionar que debido al amplio abanico de posibilidades que se pueden encontrar en los proyectos, sería necesario que de cara a futuras mejoras se ampliaran los casos comprendidos por la aplicación. También sería necesario mejorar su rendimiento dado que en proyectos de gran tamaño el análisis tardaba demasiado.

6.2. Aplicación de lo aprendido

En este capítulo se realizará una relación entre los conceptos adquiridos en la carrera y los utilizados en este trabajo. Para una mejor explicación de estos conceptos se mencionarán de manera detallada que conocimientos de las asignaturas estudiadas en el grado han servido de utilidad para la realización de este trabajo.

1. Sistemas Operativos: La utilización y aplicación de comandos en entornos Unix.
2. Ingeniería de Sistemas Telemáticos: En esta asignatura se estudia la programación orientada a objetos que es fundamental para este trabajo debido a que Python está orientado a objetos.
3. Servicio y Aplicaciones Telemáticas: Asignatura base para la realización de este trabajo debido a que también se realizan aplicaciones con el framework Django.
4. Ingeniería en Sistemas de Información: Asignatura necesaria para una utilización mas avanzada de GitHub y elaboración de proyectos siguiendo la metodología TDD.
5. Desarrollo de Aplicaciones Telemáticas: En esta asignatura se realizan interfaces de usuario lo cual ha sido de gran utilidad para la parte del cliente en este trabajo.

6.3. Lecciones aprendidas

La elaboración de este trabajo me ha servido para ampliar mis conocimientos sobre:

1. Lograr realizar un trabajo partiendo desde cero, con su respectivo desarrollo e implementación y documentando todo el trabajo de manera detallada.
2. Poder hacer frente a los problemas que puedan surgir al desarrollar un trabajo, con el consecuente análisis y la búsqueda de información adicional necesaria para poder resolverlos.
3. Ampliación de conocimientos de base de datos debido a la creación de un modelo de datos que siga una estructura lógica para una aplicación que parte desde cero.

4. La utilización de tests unitarios de Python dado que estos tests se ven con menos profundidad en las asignaturas del grado y en este trabajo he necesitado llegar a comprender cuáles son los diferentes tipos de test y cómo se implementan.
5. La utilización de la herramienta \LaTeX para la elaboración de la memoria, dado que es una herramienta con la que nunca había trabajado, pero que ha resultado ser muy útil a la hora de elaborar la memoria y con la cual se obtienen buenos resultados con una fácil implementación.

6.4. Trabajos futuros

Este trabajo cumple los objetivos marcados como ya se ha explicado anteriormente, pero sería necesario llevar a cabo futuras implementaciones para ampliar el abanico de posibilidades de estudio. De tal manera que no solo funcione y cumpla los objetivos para los casos explicados anteriormente sino que sea capaz de abarcar mas casos posibles.

Este trabajo se ha centrado en los casos que se han explicado con anterioridad debido a que desde mi punto de vista pueden ser los más simples e intuitivos para el usuario y de ahí que puedan usarse con más frecuencia. Pero aún así, quedan otros muchos casos por comprender. Por eso surge la necesidad de realizar trabajos futuros en este trabajo debido a que en los diferentes proyectos que pueden ser analizados se pueden encontrar cualquiera de las múltiples posibilidades que hay para la realización de un test unitario en Python.

6.5. Valoración personal

Este trabajo ha sido muy interesante pero al mismo tiempo complicado debido a que nunca antes me había enfrentado a un trabajo utilizando estas herramientas como han sido librerías en Python y el programa ctags.

El comienzo fue algo complicado hasta que te acostumbras a utilizarlas, lo que me llevo a tener que hacer en varias ocasiones búsquedas de información más completas y detalladas para poder entender como era su funcionamiento. Además me ha servido para ampliar mis conocimientos tecnológicos así como proponerme unos objetivos los cuales se han realizado con éxito gracias a la ayuda del profesor y de estas herramientas que se han utilizado. Además

de sentir una gran satisfacción personal al haber podido completar con éxito la totalidad de este trabajo.

Apéndice A

Manual de usuario

Para poder llevar a cabo este trabajo es necesaria la instalación de Django, el cual se instala con el siguiente comando.

```
$ pip install -e django
```

Una vez se ha instalado Django se procede a la creación de un proyecto en Django con el siguiente comando.

```
$ django-admin startproject mysite
```

A continuación dentro de la carpeta del proyecto se ejecutará el siguiente comando para la creación de una aplicación para nuestro proyecto.

```
$ python manage.py startapp myapp
```

Para migrar los cambios realizados sobre el modelo de datos, se ejecutan estos comandos de forma consecutiva.

```
$ python manage.py makemigrations
```

```
$ python manage.py migrate
```

Para arrancar la aplicación se hace con el siguiente comando.

```
$ python manage.py runserver port
```

Además de Django se han utilizado otras herramientas como son git y ctags cuya instalación se realiza con estos comandos.

```
$ sudo apt-get install git
```

```
$ sudo apt-get install exuberant-ctags
```


Bibliografía

- [1] BootStrap. Web sobre bootstrap. <https://www.getbootstrap.com/>.
- [2] Css3. Web sobre css3. <https://www.ecured.cu/CSS3/>.
- [3] Ctags. Web sobre ctags. <https://ctags.sourceforge.net/>.
- [4] Django. Web oficial de django project. <https://www.djangoproject.com/>.
- [5] Git. Web oficial de git. <https://git-scm.com/>.
- [6] Github. Web oficial de github. <https://github.com/>.
- [7] Html5. Web sobre html5. <http://es.html.net/tutorials/html/>.
- [8] JavaScript. Web oficial de javascript. <https://www.javascript.com/>.
- [9] Python. Web oficial de python programming language. <https://www.python.org/>.
- [10] SQLite. Web oficial de sqlite. <https://www.sqlite.org/>.