UNIVERSIDAD
REY JUAN CARLOS

INGENIERÍA DE TELECOMUNICACIÓN + INGENIERÍA TÉCNICA DE INFORMÁTICA EN SISTEMAS

Curso Académico 2016/2017

Proyecto Fin de Carrera

MiStuRe: MINADO DE REPOSITORIOS DE ESTUDIANTES ORIENTADO AL ANÁLISIS DEL APRENDIZAJE.

Autor : Carlos González Sesmero

Tutor : Dr. Gregorio Robles

II

"[...] nada está perdido si se
tiene el valor de proclamar
que todo está perdido y hay
que empezar de nuevo [...]".
Julio Cortázar.

II

Agradecimientos

Como Milennial educado a golpe de Hispano Olivetti y refrán castellano, un servidor tiene grabado que es de bien nacido ser agradecido, y con cariño y sarcasmo, reparte unos cuantos:

Empiezo por un fuerte agradecimiento a mi madre, y recordar aquello de “madre, no hay más que una”; y menos mal que una y no tres, ¿se imaginan?

También un gran agradecimiento a Itziar, que aunque da algo de trabajillo – ¡fuu! -, ha sido un excepcional apoyo y compañía en estos especialmente complicados últimos años.

Otro agradecimiento a Marcos que durante mi tránsito universitario pasó de llamar la atención a todas horas a ser un hombrecillo de provecho que tenía sus propios proyectos y dejaba a los demás hacer los suyos.

Un agradecimiento a toda aquella familia, amistades y allegados que durante largas temporadas de estos años compaginando mis diferentes ocupaciones solo han podido disfrutar de una pequeña parte de mí, y a la vez se alegrarán de está culminación.

Por otra parte un agradecimiento a la paciencia de Gregorio.

No olvidar al sufrido y olvidado contribuyente español, cuya indirecta contribución ha sido condición necesaria que no suficiente para que este documento haya llegado a ser redactado.

Resumen

En el contexto actual de las enseñanzas de Grado, con un profesorado con una doble exigencia, o con la autoexigencia de proporcionar una docencia con una metodología de evaluación continua y de forma muy dinámica a la vez del requerimiento de una intensa actividad investigadora. Se intuye como más que probable, que cualquier mejora o sistema que refuerce, optimice el tiempo disponible o incremente la frecuencia y la cantidad de información realimentada con respecto a los objetivos principales de una materia, tanto como de objetivos secundarios y habilidades asociadas a ellos entre los extremos típicos del acto educativo, sea positivo para ambos.

Por otra parte, en el contexto descrito, en disciplinas tales como la programación donde sea posible el tratamiento automático, obtención de patrones, o mediciones de los entregables donde los alumnos demuestran su destreza y evolución. Puede ser útil la aplicación de métodos o el uso de sistemas que permitan sistematizar y extraer información valiosa con más rapidez que con la metodología tradicional. Y que permita de forma más dinámica y objetiva la transmisión y refuerzo de los conocimientos y la detección de errores o hábitos a mejorar tanto a nivel individual como a nivel grupal.

En este escenario, y teniendo como referencia unas experiencias previas en unas asignaturas cursadas donde se emplearon estos conceptos, se va a intentar construir una propuesta de un sistema más automatizado y generalizable, orientado a alumnos con un nivel donde todavía están por formar parte de las destrezas y hábitos de programación y materias donde se aplique la programación a pequeñas aplicaciones orientadas al aprendizaje de otros conceptos como redes y servicios web.

Eminentemente, trataremos con un ecosistema donde los alumnos manejarán el lenguaje

Python, caracterizado por su simplicidad y alta interoperabilidad, y la herramienta colaborativa Git como elementos comunes.

Índice general

Índice de figuras

Capítulo 1

Introducción

En este capítulo inicial, vamos a proceder con la presentación del Proyecto de fin de Carrera que hemos denominado “MiStuRe: MINADO DE REPOSITORIOS DE ESTUDIANTES ORIENTADO AL ANÁLISIS DEL APRENDIZAJE”.

Con posteridad a la presentación, desentrañaremos la motivación personal con la que se empezó este proyecto, y describiremos la estructura de este documento, de modo que el lector tenga un mapa general de aquello que se va a encontrar a continuación, y adecue su lectura según su interés.

1.1. Presentación

MiStuRe es, en primer lugar un acrónimo de MIning STUdent REpositories, que guarda semejanza gráfica o sonora con los términos español mistura e inglés mixture.

Misture representa a un proyecto de módulos desarrollados en el lenguaje Python en su versión 3, que intenta sintetizar una estructura que proporcione un modelo de datos, una interfaz o interfaces, y unos módulos fácilmente extensibles que permitan la automatización de la corrección de prácticas de programación y que recojan con la mayor riqueza posible la información para facilitar que el alumno tenga reportes frecuentes e instruyentes acerca de su desempeño y progresión, además de poder utilizarse sistemáticamente para reportes o análisis posteriores.

La utilidad del proyecto que tiene estas aspiraciones, se engloban dentro de un ámbito educativo universitario. En el que se desea poder realizar por medio de *scripting* estas tareas de forma rápida automáticamente, y facilitar el análisis que permita detectar eficazmente los errores de los alumnos, e incluso de permitir incidir en otras destrezas del alumno, como la habilidad programando en cierto lenguaje, con cierto orden, simplicidad o calidad.

En este análisis se considera la ejecución de pruebas sobre el código o de comprobaciones sobre los ficheros entregables, para evaluar que cumplen con la funcionalidad pedida o que demuestren que la tarea ha sido efectuada correctamente, lo que implica en general un buen aprovechamiento de la actividad por parte del alumno.

Asimismo, se consideran otros análisis o comprobaciones, tales como el análisis de trazas de Wireshark, útil para el caso de la corrección de actividades relacionadas con el uso o análisis de redes, protocolos o servicios web o multimedia.

Otro de los análisis de interés, es el del uso de la herramienta Git, que en primera instancia se puede aprovechar para conducir al alumno a la adquisición de ciertos hábitos a la hora de manejar estas herramientas con su código fuente. Y en instancias superiores, a un posible análisis pormenorizado de los colaboradores con respecto al proyecto colaborativo.

1.2. Motivación Personal

Aunque la idea radical que llevó a la consecución de MiStuRe no es propia, sino parte de una propuesta del tutor. Personalmente no me resultó una propuesta no motivante, ni mucho menos extraña.

Durante el curso de las asignaturas de tercer y cuarto curso de Ingeniería denominadas IARO -Información Audiovisual en Redes de Ordenadores- y SAT -Servicios y Aplicaciones Telemáticas-, fuimos "conejillos de indias", aunque no los primeros, en cursar dichas asignaturas siguiendo una metodología no muy común en el resto de las materias.

Dicha metodología dejaba en cierto modo de lado la típica lección magistral de pizarra, quedando reducida en muchas fases de estas asignaturas a un ajustado en el tiempo *speech* sobre la materia tratada y posterior discusión de esta y de las dudas planteadas por los alumnos gracias al trabajo previo. El resto se basaba, en el trabajo y la visualización de un pequeño dosier de materiales, referencias y ejemplos, a menudo interactivos, presentados telemáticamente en las jornadas previas a la clase correspondiente, junto con la realización de tests online sobre la propia materia a continuación de realizar nuestro trabajo con este dosier.

La parte que está relacionada con MiStuRe, esta radicada en la vertiente práctica de estas asignaturas. Esta consistía en la realización de pequeños ejercicios de programación, incrementales, y en la corrección semiautomatizada de estos, que a cambio nos aportaba cierta información adicional sobre nuestro desempeño programando, y que probablemente, en más de un caso, nos generó curiosidad para mejorar nuestra calidad de escritura o estilo de código, por ejemplo.

En resumen, que dada la experiencia propia en el lado del alumno, me resulto ciertamente interesante pasar a formar parte de ello y tratar de mejorar las herramientas para potenciar esta metodología de impartir clases teóricas y prácticas.

Otra motivación que me llevo a adoptar la realización de un proyecto así, fueron las múltiples posibilidades de temas a profundizar o tecnologías que usar en las que aprender o ampliar el desempeño en ellas, que podía permitir la implementación de un escenario así.

1.3. Estructura de la memoria

Esta memoria, relativa al proyecto de fin de carrera denominado como MiStuRe, al cuál desde este momento denominaremos como Misture o proyecto para simplificar, consta de seis capítulos, apéndices sobre las nociones de uso de Misture y la instalación de las debidas dependencias para su correcto funcionamiento, y de un apartado de bibliografía.

En el primer capítulo, que estamos concluyendo con este subapartado, se dedica a la introducción del proyecto Misture, la motivación del autor para llevarlo a cabo, y esta explicación de la estructura que se está realizando.

En un segundo capítulo, el de objetivos, se detalla el problema a resolver con el proyecto Misture, concretamente de mejorar la funcionalidad de unos *scripts* piloto, cuya estructura se analizará en general, y en base a este punto de partida, definiremos los objetivos.

En sucesivos subapartados de este capítulo, también se detallan los objetivos perseguidos por el tutor tanto como por el autor con la realización de Misture, tanto a nivel práctico como a nivel teórico.

Asimismo, se detallarán los requisitos funcionales que se propusieron inicialmente para Misture, que constituyen en sí mismos otro objetivo.

En el capítulo tercero, denominado “Estado del Arte”, nos centraremos en detallar aquellas tecnologías, librerías o herramientas clave, así como unas reseñas sobre los “Mining Software Repositories”.

En el capítulo cuarto, se describirá el diseño y arquitectura general de Misture, de sus componentes, de que elementos y utilidades nos servimos en su implementación, la estructura de las BBDD utilizadas.

El penúltimo capítulo, “Resultados”, se resume que objetivos se han intentado cumplir finalmente y en que grado se han cumplido o en que estado han quedado.

La cuenta de capítulos queda cerrada con el de “Conclusiones”, donde recopilamos las ideas y resoluciones a las que hemos llegado con la elaboración de este proyecto.

Tras el desarrollo de la memoria, adjuntamos los apéndices con aclaraciones de uso y de las particularidades de las dependencias necesarias para la ejecución del proyecto, y la bibliografía las principales referencias utilizadas a lo largo del desarrollo de Misture.

Capítulo 2

Objetivos

2.1. Descripción del problema

El problema al que se quiere dar una solución es la construcción de un sistema, que a partir de unas configuraciones de entrada, y cuya especificación sea trasladable al enunciado de las actividades prácticas de alumnos de asignaturas de programación de servicios y aplicaciones sobres redes de ordenadores, pueda proceder a su corrección requiriendo los mínimos cambios o ajustes en la implementación en dicho sistema.

Estas correcciones es deseable que puedan realizarse lo más desatendida y automáticamente posible, permitiendo liberar tiempo al profesor y disminuir el tiempo de respuestas, además de proporcionarles con poco retraso con respecto al día de entrega de un conjunto variado de información acerca de los resultados de sus prácticas con el objetivo de incidir positivamente en el aprendizaje tanto de los conceptos de la asignatura como de habilidades en programación.

Los elementos que debemos considerar en la corrección del alumno son los siguientes:

- En general, un repositorio Git, no “bare”, en el cual el alumno ha estado trabajando y pueda observarse su evolución trabajando en la actividad, cuyo directorio de trabajo contenga los entregables predefinidos por la especificación o enunciado de la actividad.

- Ficheros de código fuente: trabajando casi siempre con fuentes en Python 2 o 3.

- Capturas de Wireshark: dado que estamos evaluando a alumnos que cursan asignaturas relacionadas con servicios o aplicaciones lanzadas sobre redes, es un tipo de entregable muy útil para evaluar el funcionamiento de programas cliente o servidor que se entregan.

- Otros ficheros de texto o XML, cuya inclusión venga justificada por contener configuraciones o datos de entrada para los *scripts*, o sean en sí mismo uno de los entregables evaluables, como por ejemplo una actividad acerca del lenguaje SMIL, que es una especificación XML.

En base a esos entregables, tenemos que plantearnos el sistema automatizado, que nos ayude a evaluar con el mayor grado de detalle posible que la actividad ha sido entregada completamente en los términos especificados por el enunciado y la funcionalidad se cumple, demostrando el aprovechamiento de la actividad.

2.2. Punto de partida

Para la realización de este sistema, no se parte de cero. Ya que fruto de las nuevas experiencias docentes introducidas por el profesor, éste desarrolló unos *scripts* semiautomáticos para poder agilizar la corrección y disponer los alumnos de sus reportes antes de la siguiente sesión práctica.

Aunque estos *scripts* todavía exigen bastante intervención manual , es una muy buena primera aproximación y permite abordar todo el proceso de corrección que de otra forma llevaría días o semanas, y sin tanta cantidad de información y resultados susceptibles de evaluar y poder incidir sobre ellos en las clases.

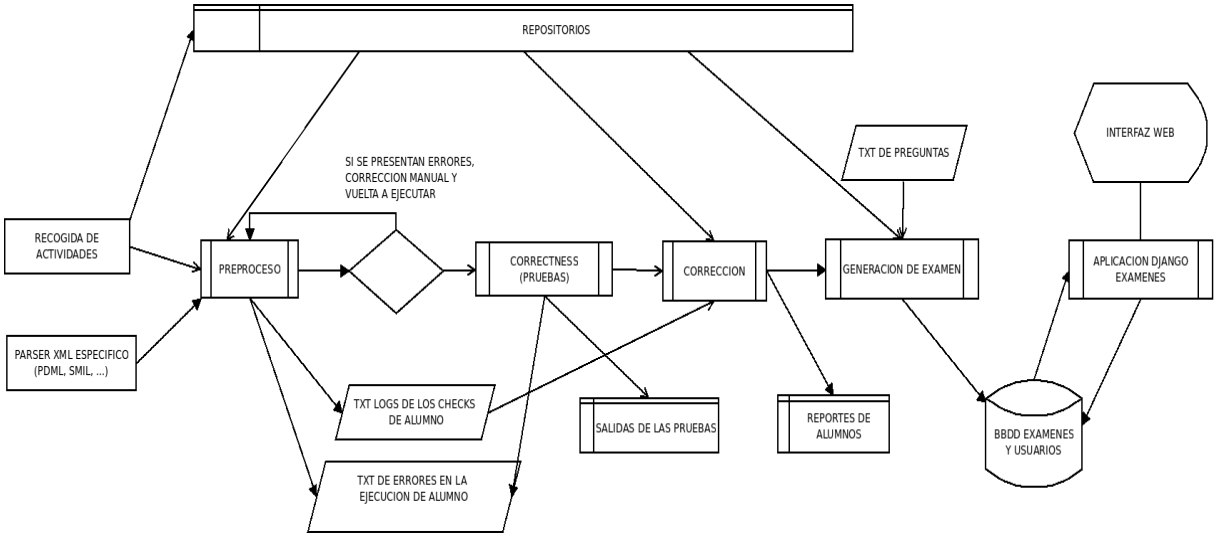


Figura 2.1: Flujo de los scripts semiautomáticos de la experiencia piloto

Según algunos reportes del profesor, con esta metodología, se podía evaluar una actividad en una mañana o una jornada, según la complejidad particular de las correcciones, y teniendo en cuenta muchos más información disponible para evaluar que otras formas de evaluación prácticas habituales en asignaturas similares.

Este sistema de *scripts*, que se introduce en la figura anexa, se compone de los siguientes procesos, la mayoría de ellos implementados por un módulo o script Python.

2.2.1. Recuperación de los datos

En general es un proceso aparte de los *scripts* planteados. Cuando yo realicé estas experiencias, consistía en que los administradores de los laboratorios de la universidad automáticamente recopilaban las rutas de entrega que se habían especificado dentro del *home* de cada alumno.

Actualmente, se solicita el usuario público de los alumnos en el servicio Github, donde se especifica que inicien un repositorio con el nombre especificado para realizar la entrega, para configurarlos en el preproceso y clonarlos.

2.2.2. Preproceso

En este módulo se lee y establece la configuración inicial con las especificaciones de la entrega tales como los usuarios de los alumnos, localización de repositorios, listado de ficheros a entregar, etc.

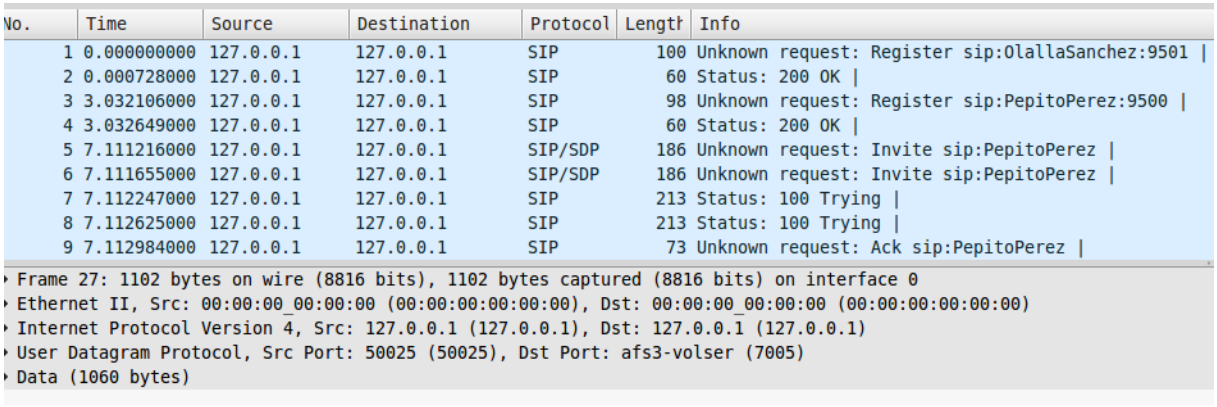
Una vez establecida la configuración. Se lanzan diferentes comprobaciones sobre los ficheros mediante funcionalidad python o invocando terceras utilidades con llamadas al sistema. La salida se vuelca a ficheros de texto en un formato parseable, y los errores en la ejecución de los "checks" a otro fichero de errores, que revisa el profesor por si tiene que tomar acciones correctoras sobre el repositorio afectado y ejecutar nuevamente los "checks".

- La existencia del listado de ficheros exigido.
- Evaluación de errores de estilo en el código Python, con ayuda de pep8.
- Evaluación de prácticas de mala calidad de código, con pylint.
- Lectura del *log* del repositorio Git, extrayendo el número de commits y su fecha.
- En las actividades que lo exijan, ejecutar el módulo parser XML personalizado a la actividad para el chequeo de capturas Wireshark, convertidas a PDML, o la evaluación del contenido de las etiquetas y los atributos de cualquier otro fichero XML.

2.2.3. Parseo XML: PDML, SMIL

La utilidad de este módulo sale a relucir en las entregas dónde los alumnos entregan capturas Wireshark con las trazas resultantes de ejecutar sus escenarios, las cuáles se convierten a PDML (ver las figuras inferiores) para facilitar su análisis. También en alguna otra actividad

donde se han tratado con lenguajes XML, como el caso de SMIL.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	SIP	100	Unknown request: Register sip:OlallaSanchez:9501 |
2	0.000728000	127.0.0.1	127.0.0.1	SIP	60	Status: 200 OK |
3	3.032106000	127.0.0.1	127.0.0.1	SIP	98	Unknown request: Register sip:PepitoPerez:9500 |
4	3.032649000	127.0.0.1	127.0.0.1	SIP	60	Status: 200 OK |
5	7.111216000	127.0.0.1	127.0.0.1	SIP/SDP	186	Unknown request: Invite sip:PepitoPerez |
6	7.111655000	127.0.0.1	127.0.0.1	SIP/SDP	186	Unknown request: Invite sip:PepitoPerez |
7	7.112247000	127.0.0.1	127.0.0.1	SIP	213	Status: 100 Trying |
8	7.112625000	127.0.0.1	127.0.0.1	SIP	213	Status: 100 Trying |
9	7.112984000	127.0.0.1	127.0.0.1	SIP	73	Unknown request: Ack sip:PepitoPerez |

```
▸ Frame 27: 1102 bytes on wire (8816 bits), 1102 bytes captured (8816 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▸ User Datagram Protocol, Src Port: 50025 (50025), Dst Port: afs3-volser (7005)
▸ Data (1060 bytes)
```

Figura 2.2: Traza de una captura de red visualizada

La aproximación que se sigue para el análisis, es utilizar el módulo SAX de Python para implementar específicamente para la actividad concreta que estamos corrigiendo, las comprobaciones sobre estos ficheros sobre las etiquetas, atributos o valores de ellas. Para demostrar que la traza o el XML ha sido generado o tiene los contenidos que se esperan, de haber realizado el alumno la tarea correctamente.

```
  <proto name="sip" showname="Session Initiation Protocol (Register)" size="58" pos="42">
    <field name="sip.Request-Line" showname="Request-Line: Register sip:OlallaSanchez:9501 SIP/2.0" size="39" pos="42"
           show="Register sip:OlallaSanchez:9501 SIP/2.0" value="52656769737465722073697703a4f6c616c6c6153616e6368657a3a39353031
      <field name="sip.Method" showname="Method: Register" size="8" pos="42" show="Register" value="5265676973746572"/>
      <field name="sip.r-uri" showname="Request-URI: sip:OlallaSanchez:9501" size="22" pos="51" show="sip:OlallaSanchez:9501" va
        <field name="sip.r-uri.host" showname="Request-URI Host Part: OlallaSanchez" size="13" pos="55" show="OlallaSanchez" val
        <field name="sip.r-uri.port" showname="Request-URI Host Port: 9501" size="4" pos="69" show="9501" value="39353031"/>
      </field>
      <field name="sip.resend" showname="Resent Packet: False" size="0" pos="42" show="0"/>
    </field>
    <field name="sip.msg_hdr" showname="Message Header" size="17" pos="83" show="Expires: 3600\x0d\x0a\x0d\x0a" value="457870697
      <field name="sip.Expires" showname="Expires: 3600" size="15" pos="83" show="3600" value="457870697265733a20333630300d0a"/>
    </field>
  </proto>
</packet>
```

Figura 2.3: Extracto de un PDML de una trama SIP dentro la última capa de un paquete

Una vez el profesor ha implementado las comprobaciones en un módulo *parser* específico, se invoca en el módulo de preproceso y se recogen los resultados impresos en el *log* desde el módulo de corrección.

2.2.4. Correctness

Correctness es el módulo de ejecución de pruebas de caja negra, se lanzan las llamadas sobre los *scripts* de los alumnos para probar su funcionalidad ante determinadas entradas o ficheros de entrada, y se recogen los resultados y errores en los *logs*.

Estas salidas se pueden comparar en el posproceso con el resultado esperado por el procesor, en el caso de actividades más sencillas o aquellas que se defina en el enunciado un formato de salida muy estricto. En caso contrario, la salida se almacena dentro del pool de ficheros de reporte del alumno para la posterior inspección del profesor.

2.2.5. Posproceso

La funcionalidad de este *script* es parsear los logs del preproceso con el resultado de los "checks" y , generando un reporte en formato texto con el detalle sobre el resultado de las comprobaciones realizadas y su resultado, y el desglose de sus errores de estilo y avisos sobre líneas de código que incurren en algún uso del código no considerado de calidad según lo generalmente aceptado. Según el caso, también el resultado de las pruebas de caja negra.

2.2.6. Generación de examen

En este script, se inyecta a una base de datos SQL, asociada también a una aplicación Django, las preguntas de la prueba que se hace a los alumnos para evaluar si el alumno conoce su propio código y cuestiones que evaluen.

Estas preguntas se inyectan de dos fuentes. La primera viene desde los propios ficheros fuente de la entrega, para la cual se inyecta un fragmento de ćodigo y se pregunta a cuál de los ficheros fuente pertenece.

La segunda fuente, es un fichero de texto, parseable, donde el profesor formula preguntas acerca de conceptos aplicados en la actividad, o cuestiones donde se le pide al alumno que indique cuál es la salida o que cambio tiene que hacer en un fragmento de código indicado.

```
Nuestro proxy/registrar recibe el siguiente mensaje (estando "sara" registrada):

<pre>
INVITE sara

o=iker@realmadrid.com 127.0.0.1
s=A mi me gusta el pipipiripiriii
t=0
m=audio 12345 RTP
</pre>

¿qué responderá tu servidor proxy/registrar? (código numérico y respuesta textual)

*****400 Bad Request
```

Figura 2.4: Formato de la pregunta de examen que se sube a la BBDD

2.2.7. Detección de plagio

Esta tarea no está integrada dentro del resto de scripts, pero también forma parte del proceso de corrección, especialmente en las últimas entregas de las diferentes asignaturas.

En este paso del proceso de corrección, se suben las rutas con los códigos al servidor de la herramienta MOSS, donde se realiza el análisis y pasado un tiempo devuelven.

MOSS mide la similitud de entre ficheros texto a través de métodos estadísticos y establece esta medida fichero a fichero. Según el tipo y no es un hecho fuera de lo habitual encontrarse ante falsos positivos, especialmente si usamos MOSS para cotejar los códigos de las actividades más simples.

File 1	File 2	Lines Matched
asalvati/calcoohija.py (96%)	rodrigu/calcoohija.py (96%)	45
camichan/calcoohija.py (54%)	camichan/calcoo.py (59%)	37
asalvati/calcoo.py (92%)	rodrigu/calcoo.py (90%)	39
perez/calcoo.py (91%)	sblazque/calcoo.py (86%)	36
agrasa/calcoohija.py (66%)	agrasa/calcoo.py (76%)	35
laualva/calcoohija.py (82%)	rzazo/calcoohija.py (88%)	34
alex/calcoohija.py (80%)	rzazo/calcoohija.py (88%)	32
amaitia/calcoo.py (95%)	rzazo/calcoo.py (95%)	44
perez/calcoohija.py (70%)	sblazque/calcoohija.py (66%)	29
dbutragu/calcoo.py (97%)	leh/calcoo.py (97%)	26
alex/calcoohija.py (73%)	laualva/calcoohija.py (75%)	26
dbutragu/calcoohija.py (74%)	pfernand/calcoohija.py (46%)	30
amaitia/calcoohija.py (72%)	rzazo/calcoohija.py (72%)	30

Figura 2.5: Resultado que publica MOSS en una url

2.3. Requisitos

Una vez estudiado todo el escenario de nuestro problema y la funcionalidad semiautomática de los *scripts* de corrección, se estuvo definiendo de forma general entre proyectando y tutor algunos de los requisitos que se extrajeron de este análisis, obteniendo así el siguiente listado de requisitos a tener en cuenta en el sistema Misture:

- Análisis de qué ficheros han sido entregados en el repositorio, mejorando la experiencia del *script*, limitada a verificar un listado inicial de ficheros.

- Solucionar el caso frecuente de pequeñas desviaciones en los nombres de fichero.

- Identificación del tipo de código de una fuente o el tipo del fichero, y algunos parámetros adicionales como la cantidad de líneas de código.

- Análisis de la estructura de los códigos Python ampliado.

- Incrementar la funcionalidad inicial que se limitaba a contar las clausulas “class” o “def” en los ficheros Python determinados inicialmente.

- Análisis de errores de estilo, y calidad de código. Los pilotos cuentan la cantidad de errores de estilo de pep8 y de warnings sobre un análisis estático del código de pylint y reportan al profesor un resumen en formato texto de la cantidad de errores por tipo.

- Enriquecer la funcionalidad del análisis Git, hasta ahora basado en el análisis del *log* del repositorio, contando la cantidad de commits y la marca de tiempo entre primer y último commit.

- Ampliar la funcionalidad del tratamiento de las capturas Wireshark, aprovechando todas las posibilidades que permite la conversión a PDML.

- Como un plus, plantear, de ser posible, una utilidad de antiplagio libre como sustitutiva a MOSS.

2.4. Objetivos generales

Una vez descrito el problema con el que estamos tratando, partiendo de la base de las experiencias piloto con la parte práctica de las materias, podemos establecer cuál es el objetivo.

El objetivo principal de Misture, es la integración de un conjunto de utilidades y de diferentes comprobaciones y pruebas de diferente índole, que van a enfrentarse contra el conjunto códigos fuentes y ficheros entregables que los alumnos generan en las diferentes actividades.

Esta integración debe buscar que el proceso sea lo más parecido a un flujo automatizado que reduzca el tiempo empleado por el corrector, que le permita centrarse en aquellas subtareas de una corrección donde más pueda aportar.

Por ejemplo, el profesor en base a estadísticas del análisis y a indagar a partir de ellos, puede detectar ciertos hábitos o malas prácticas desarrollando o aplicando los conceptos teóricos de la asignatura, o decidir dar una clase de refuerzo, enseñar cierta práctica o patrón de diseño, etc. Y esto es más valioso para todos que si tuviera que centrarse más bien en chequear entrega a entrega si un alumno ha escrito más o menos funciones, de más o menos longitud o complejidad, si su forma de escribir código es poco clara, etc.

Asimismo, el objetivo incluye un tratamiento más organizado de la información extraída, por ejemplo en BBDD, y la posibilidad de poder ofrecer un reporte más rico, sencillo, y fácil de ampliar o modificar frente al tratamiento que ofrecen los ficheros txt, que contienen las salidas heterogéneas de las diferentes utilidades y pruebas.

Para la consecución de dicho objetivo, tenemos como punto de partida el conjunto de scripts semiautomáticos que nos ha proporcionado el tutor utilizadas para las experiencias piloto, junto al reporte de incidencias u observaciones realizadas durante la ejecución y realización de alguna corrección, para intentar buscar mejoras en las funcionalidades que permitan reducir los puntos críticos en la corrección, que incapacitan la acción del *script* y llevan a una revisión manual por parte del tutor.

2.5. Objetivos Específicos

Aunque ya definidos los requisitos funcionales generales que exige el problema, y los objetivos generales. De estos podemos derivar algunos y concluir otros complementarios, que no debemos perder de vista para la consecución del objetivo general:

- Establecer el proyecto Python, que recogiera las funciones básicas, detectadas en las

muestras de las actividades entregables de las asignaturas de programación de aplicaciones y servicios Web y multimedia en redes, adaptada a las características de las pequeñas prácticas realizadas por los alumnos.

- Encontrar los mecanismos necesarios, para automatizar al máximo cada una de las funcionalidades básicas, reduciendo al mínimo la cantidad de pasos o funcionalidades semiautomáticas o manuales llevabas a cabo habitualmente por el profesor en las experiencias piloto descritas.

- Como en nuestro contexto nos hallamos ante alumnos en proceso de aprendizaje de destrezas de programación. Se debe buscar el permitir cierta tolerancia en la automatización a errores por parte de los alumnos en los códigos o en los ficheros entregables.

  Este objetivo puede en muchos casos añadir complejidad a la resolución de nuestros propósitos. Pero la detección y rectificación automatizada de estos redundaría en reducir las intervenciones manuales y re-ejecuciones de los *scripts* de los diferentes pasos de la corrección.

- En consecuencia a los anteriores, disminuir el tiempo dedicado por el docente en el desarrollo o adaptación del código para la corrección de nuevas actividades, su ejecución, y la recogida de resultados y reporte de los puntos críticos a revisar.

- Mejora en la generación, riqueza, y comunicación del *reporting* de los resultados de los diferentes análisis y correcciones que se ejecutan sobre los repositorios.

- Mejorar la recogida de información extraída en las correcciones, de forma persistente y debidamente relacionada en sistemas de BBDD, permitiendo nuevos análisis posteriores, y ahorrando tiempo y re-ejecuciones en caso de errores que requieran alguna intervención

manual por parte del profesor.

- Interrelacionar nuestro sistema con una interfaz web, en una primera aproximación, para la realización de exámenes de verificación de autoría y conocimiento del código del alumno. Asimismo, mejorar la experiencia piloto de forma que se pueda extender la funcionalidad de la interfaz web a mostrar.

- A título personal, con la realización de este proyecto, pretendía, en el momento de su elección, adquirir el conocimiento o destreza en diferentes áreas y tecnologías:

  - Python 3, puesto que durante mi ciclo universitario la versión generalmente estudiada era Python 2.

  - MongoDB, cuyo paradigma de BBDD no había tenido la oportunidad tampoco de estudiar y practicar en ninguna asignatura.

  - Git, del cuál tenía un breve conocimiento básico – add, commit, pull, push – y empleado para usos individuales y no colaborativos.

  - Ampliar conocimientos sobre el análisis de código fuente, de repositorios, de fuentes de información heterogéneas.

Capítulo 3

Estado del arte

En este capítulo se describen las tecnologías utilizadas para la realización de la aplicación.

3.1. OTRAS HERRAMIENTAS UTILIZADAS

El desarrollo de este proyecto se ha realizado eminentemente desde y para sistemas Linux, y concretamente se ha desarrollado y probado en distribuciones Ubuntu y Linux Mint, semejantes a los entornos que emplean profesores y alumnos en el contexto tratado.

No obstante, dado que Python es un lenguaje soportado por otros sistemas, y que en la iteración más reciente del desarrollo del proyecto se ha tendido a emplear las utilidades a través de librerías o APIs Python, o son herramientas con versión en otros sistemas. En un gran porcentaje sería portable este proyecto a otros sistemas como Windows.

En todo caso, inicialmente el desarrollo no esta considerado para otros sistemas, y exigiría la revisión de la totalidad de las dependencias de las librerías utilizadas. También sería necesaria la revisión y refactorización de algunos códigos, como por ejemplo aquellos que emplean la librerías estándar *os* y *os.path*, para asegurar la plena independencia del código frente al sistema operativo.

Una vez circunscritos a estos sistemas, hay que recalcar que en general se ha intentado optar por el uso de editores, herramientas, librerías, DBMS de software libre, open source o en todo caso amigables con sus filosofías o licencias compatibles.

Aún así, durante el transcurso del proyecto, pasé a utilizar como IDE la herramienta Pycharm, que es privativa en su versión profesional, aunque contando con licencias especiales para proyectos de open source, y una versión básica con licencia no privativa.

En este caso, el cambio a este IDE viene justificado con una notable mejora de productividad ayudando a manejar la cantidad creciente de código, las ayudas y atajos para recordar y acceder a todas las librerías y APIs que he ido integrando, una gran ayuda en la detección de errores y *warnings* previos a la ejecución, y en ejecución gracias a un potente debugger.

Una vez ya han sido introducidas con anterioridad las tecnologías, herramientas y librerías clave, paso a hacer una mención de dos de las herramientas que he conocido durante la realización de este proyecto, y que han sido de notable ayuda para su realización.

3.1.1. PYCHARM

Como ya se ha introducido, PyCharm es un IDE multiplataforma orientado al lenguaje Python, y también al desarrollo de JavaScript y soporte para frameworks basados en estos lenguajes cono AngularJS o Django.

Pycharm aporta además, entre otras funcionalidades:

- Ayuda y análisis de código: autocompletado, marcado de sintaxis y errores.
- Navegación avanzada entre los elementos del proyecto y del código.
- Refactorización del código Python.
- Depurador integrado.

- Integración para test unitarios en Python.

- Integración con sistemas de control de versiones: Git, Mercurial, Subversión, CVS.

- Extensiones y disponibilidad de una numerosa biblioteca para infinidad de propósitos.

- También permite la integración de herramientas externas. Por ejemplo, en nuestro caso integramos *pep8* y pudiéndose chequear el estilo de cualquier módulo rápidamente.

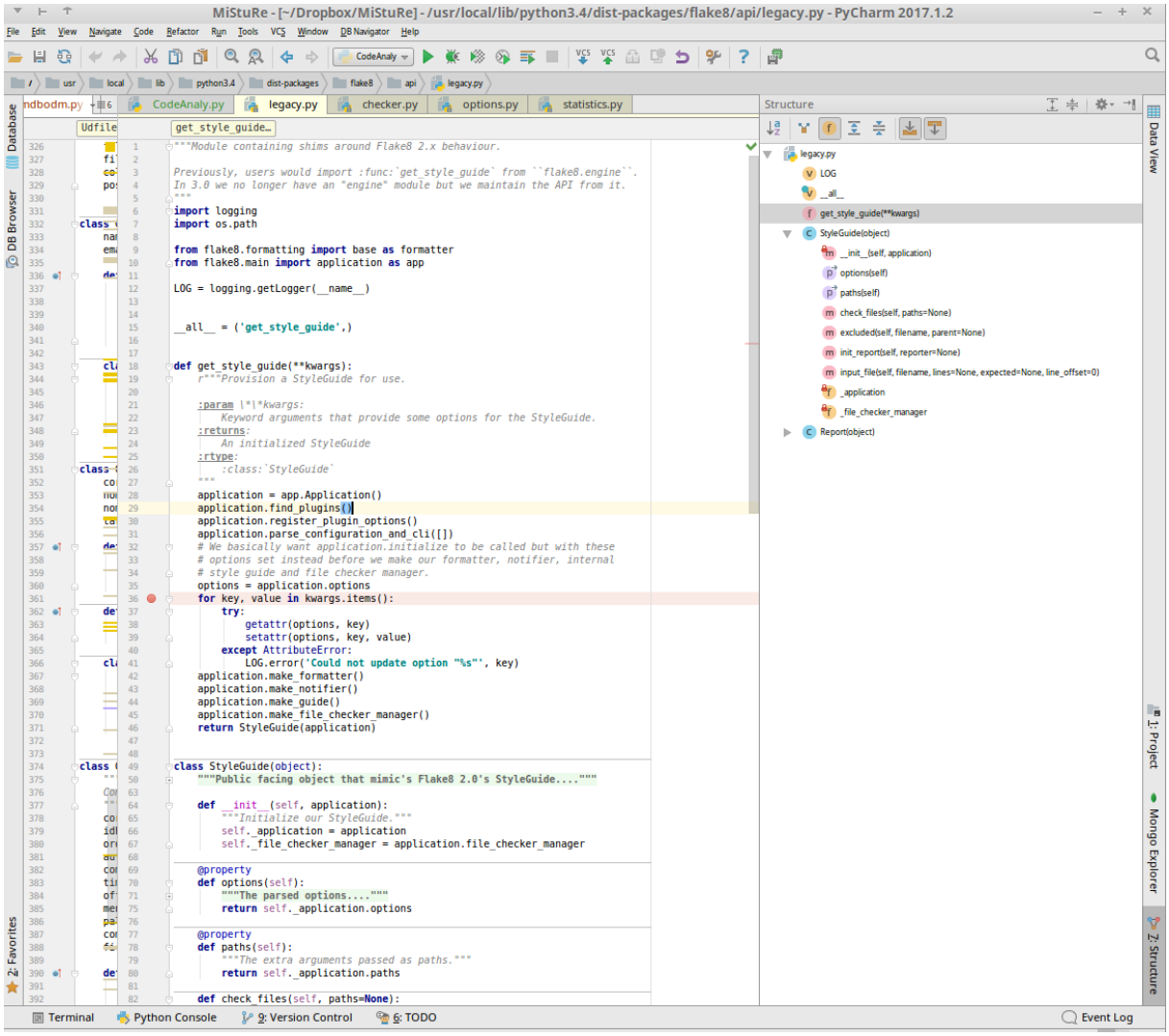


Figura 3.1: Interfaz de PyCharm

Jetbrains, la empresa desarrolladora de este producto, lo pone a disposición de los usuarios bajo un modelo que puede considerarse *fremium*. Tienen disponible una versión profesional, privativa, con todas las funcionalidades y extensiones.

No obstante, proporcionan licencias especiales para proyectos open source no comerciales, cuyas condiciones podemos encontrar en `https://www.jetbrains.com/buy/opensource/`.

Asimismo, mantienen versiones básicas de sus productos lanzadas bajo licencias open source, como Apache 2.0, llegando a disponerse públicamente del código de algunos de ellos. Tal como es el caso de IntelliJ IDEA `https://github.com/JetBrains/intellij-community`, que es otro IDE desarrollado por ellos que es la propia base del IDE PyCharm.

En el caso de PyCharm, está versión es la denominada como “Community Edition”, la cuál fue la que se empleo en el desarrollo y depuración de buena parte de este proyecto.

3.1.2. ROBOMONGO

Robomongo es un cliente para bases de datos MongoDB con interfaz gráfica, muy ligero y de funcionalidad básica, pero suficientemente potente.

Además de integrar la funcionalidad del cliente shell de MongoDB, permite la escritura de scripts para mongo y la visualización rápida de las diferentes BBDD MongoDB de la conexión sobre la que se establece, asimismo de las principales características, configuraciones y contenidos de colecciones y documentos.

Una característica simple pero que resulta de gran ayuda, es la rapidez de navegación entre las diferentes colecciones y documentos y la posibilidad de visualización de estos en diferentes formatos como tabla, árbol, o la tradicional vista de texto en formato JSON.

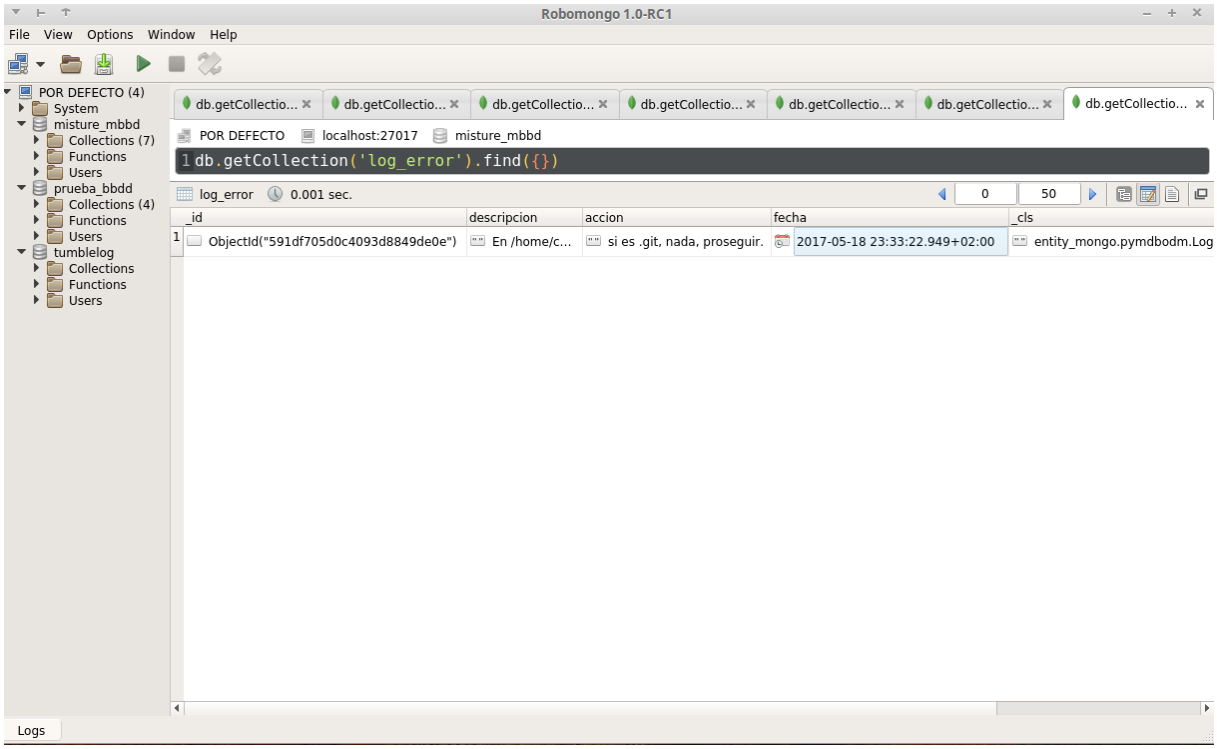


Figura 3.2: Interfaz de Robomongo

Robomongo ha estado siendo un proyecto open source cuyo código ha estado disponible públicamente. Recientemente, ha sido adquirido por la empresa Studio3T, que ha optado por mantener el proyecto en open source `https://github.com/Studio3T/robomongo`, en paralelo a sus proyectos comerciales relativos a MongoDB.

Capítulo 4

Diseño e implementación

En este capítulo se realiza una descripción detallada sobre el diseño y la implementación del presente Proyecto de Fin de Carrera.

4.1. Modelo de desarrollo

El desarrollo de un proyecto de software libre implica una sucesión de tareas entre el momento que se tiene una idea para resolver un problema o necesidad, y el producto u servicio final que lo satisface. Este metodología nos marca como se suceden las diferentes tareas y actividades dentro del proyecto.

Durante el desarrollo de este proyecto hemos seguido un modelo que se asemeja al modelo de desarrollo iterativo y creciente, o incremental. En este proceso creamos una primera versión del sistema, funcional, con el que se pueda interactuar y nos dé realimentación para las sucesivas iteraciones.

El modelo incremental, a priori, nos proporciona las siguientes ventajas:

- Al desarrollar sistemas más pequeños con subconjuntos de los requerimientos o funcionalidades, reducimos el riesgo asociado a realizar el desarrollo en bloque del sistema

completo.

- Al desarrollarse progresivamente parte de las funcionalidades, se puede evaluar mejor si los requerimientos de los siguientes incrementos son correctos o hay que adaptarlos a la realidad.

- En caso de errores o problemas, generalmente afecta a la última interacción y no a todo el conjunto, volviendo al incremento previo en el peor caso.

- En el caso de este proyecto, me permitía avanzar paso a paso en el desarrollo del sistema mientras iba adquiriendo la experiencia necesaria para desarrollar mejor o redefinir mejor los siguientes incrementos.

Además, este modelo tiene gran similitud con el modelo que nuestro profesor adoptaba para la evolución de las prácticas de sus asignaturas durante el curso. Las cuales empezaban con prácticas sencillas o introductorias, a las que se le iban añadiendo elementos, funcionalidades y complejidad, hasta llegar a la práctica final, que mirando retrospectivamente, había sido una sucesión de incrementos a partir de una estructura inicial que se había establecido prácticas atrás.

Para observar esto último, en el caso de este proyecto, disponíamos de algunas muestras de grupos de repositorios de actividades en diferentes etapas del trimestre, los cuales, por ejemplo, podemos clasificar según a la exigencia que nos va a suponer para nuestro sistema corrector Misture.

GRUPO 1

- Actividades en las semanas iniciales del trimestre o introductorias.

- Pequeños proyectos con poca cantidad de código y elementos que exige un análisis de

código Python más simple.

- Salidas de ejecuciones más sencillas: cierto fichero con un contenido concreto, cadenas cortas de valores concretos, etc.

- Puede requerir algunas comprobaciones sencillas sobre ficheros o documentos XML

- Por el momento, al alumno se le está introduciendo en temas de calidad y estilo de código, y es suficiente con contar los errores e imprimirle al alumno que errores comete, que significan y en que código se produce.

GRUPO 2:

- Actividades de etapas avanzadas del trimestre o de materias de mayor nivel.

- Escenarios de mayor cantidad y complejidad de código fuente: aplicaciones cliente-servidor con lógicas complejas, desarrollo de aplicaciones sobre frameworks que conllevan mayor complejidad en la estructura del proyecto entregado, etc.

- Análisis de código Python más complejo y rico.

- La propia prueba de ejecución de la aplicación o aplicaciones se vuelve más difícil, y no nos sirven de igual forma cierto tipo de pruebas de ejecución en unos escenarios u otros.

- En este punto, es interesante obtener información más rica sobre estilo y calidad de código y errores, para poder luego analizarse o reportarse asociado a su fichero fuente, u observar la evolución temporal de la calidad y errores de nuestro código.

De acuerdo al modelo de desarrollo descrito, en una fase inicial, recogimos de forma general los requerimientos del sistema completo, en base a la descripción del problema y las experiencias previas descritas en el Capítulo 2 de la presente memoria.

Una vez se disponen de los requisitos, se crea una versión inicial funcional, que comenzamos con un modulo principal, que disponía de las configuraciones básicas para obtener los repositorios y conocer los requisitos básicos como que ficheros deben entregarse.

A partir de ese versión inicial, se fueron añadiendo los módulos con los diferentes grupos de funcionalidades, los cuáles íbamos probando frente a las muestras de los repositorios de alumnos de las que dispusimos. Es decir, realizábamos incremento para agregar funcionalidad básica de análisis de código Python, posteriormente otra iteración para la extracción de información básica del repositorio git, etc.

Para simplificar la explicación del diseño de Misture, vamos a reducir la explicación de todo el proceso al detalle de dos estados, que vamos a denominar iteración 1 e iteración 2, que realmente componen una sucesión de incrementos que llevaron a esos dos estados del proyecto.

Esta división, asimismo, tiene también como justificación el hecho de que el estado descrito en la primera iteración es más próxima a las actividades caracterizadas del grupo 1 descrito anteriormente, y la iteración 2 recoge un estado más cercano a resolver las necesidades de las actividades descritas en el grupo 2.

4.2. ITERACION 1:

Vamos a describir el diseño de los diferentes módulos que componen este primera sucesión de incrementos. En la figura de abajo se muestra un esquema de flujo de está primera “versión” de Misture.

En este diseño, para el almacenamiento y tratamiento de todos los datos extraídos en los diferentes análisis, se emplearon ficheros para las transacciones entre las utilidades externas y

para generar logs de errores e imprimir el reporte final ordenado, así como estructuras dinámicas como listas y diccionarios y una jerarquía de clases Python que modelaban los diferentes objetos y las relaciones entre ellos que identificábamos en este escenario, tales como actividad, alumnos, repositorios, ficheros, elementos de código, estadística, error de estilo, etc.

Esta jerarquía, era una primera versión del modelo usado en la iteración 2 y que se detalla posteriormente en en el apartado “DISEÑO DE BBDD”.

4.2.1. MÓDULO PRINCIPAL

En el modulo MainAnaly, se establece la configuración básica de la actividad a corregir, como la ubicación de los repositorios asociados a los logines de alumnos, los nombres de los ficheros y códigos fuentes que deben estar en el repositorio, así como las tuplas con las llamadas a los programas del alumno y las salida que debe obtener el programa en caso de funcionar correctamente según la especificación de la actividad.

Este módulo, también se encarga de cargar esta configuración en la jerarquía de clases del escenario e inicializar los logs para la actividad y establecer una ubicación para los reportes individualizados.

A continuación, realiza las diferentes llamadas a los módulos de análisis, a los que les proporciona como parámetros los objetos adecuados del escenario -repositorio, fichero, etc- y recoge los resultados del análisis, asociándolos al objeto del mundo Misture correspondiente.

4.2.2. MÓDULO DE ANÁLISIS DE CÓDIGO PYTHON

Este módulo, llamado *PyfichAnaly*, implementa la funcionalidad relativa a la identificación de los ficheros presentes en el repositorio del proyecto, en función de una lista predefinida de ficheros.

Como novedad, en este análisis se detecta si al alumno le falta alguno de los ficheros requeridos por la práctica, y a través de una pequeña heurística intenta deducir si hay otros ficheros

que de nombre similar que puedan serlo.

Esta heurística, se basa en el algoritmo de Levenhstein, es un algoritmo que tiene la capacidad medir la cantidad de operaciones de inserción, borrado o cambio de caracteres que hay entre dos cadenas de texto.

Por tanto, en caso de que se encuentren archivos huérfanos en la lista de ficheros requeridos y en el repositorio, y siempre que no supere un umbral, se entiende que el fichero con la menor distancia de Levenhstein es aquel que nos falta, y por tanto, se renombra y se prosiguen los análisis.

Una vez identificados los ficheros disponibles, se identifican los que son ficheros fuente Python, y se llama a la clase **PyModuleCont**, que analiza el código fuente del fichero a través de patrones regulares y funciones de las librería re de Python, cuya muestra se adjunta en la figura inferior.

De dicho análisis, asociamos al fichero Python los elementos clase, método y función que contengan, junto a su nombres.

```
NAMEPAT = r"\s*(?P<" + name + ">[_A-Za-z][_A-Za-z0-9]*)\s*"
PARAMPAT = r"\s*(?:[_A-Za-z][_A-Za-z\d]*){0,1}\s*"
CNTLINE = r"\s*\\\n" # '   \\n'
CLASSPAT = r"^class\s+" + namerPatvars('class') + r"(?:\([^\(]*\)){0,1}\s*:\s*"
FUNCPAT = r"^def\s+" + namerPatvars('function') + r"(?:\([^\(]*\)){0,1}\s*:\s*"
METHPAT = r"^\s+def\s+" + namerPatvars('method') + r"(?:\([^\(]*\)){0,1}\s*:\s*"
MAINPAT = '|'.join((FUNCPAT, CLASSPAT, METHPAT))
```

Figura 4.1: regex utilizados para identificar elementos de código y su nombre

4.2.3. MÓDULO DE ANÁLISIS DE CALIDAD DEL CÓDIGO

En el módulo **CodeAnaly.py**, se implementa la funcionalidad de análisis de calidad del código, que en esta iteración del desarrollo de Misture, se realiza con ayuda de las utilidades *Pep8*, para el análisis de estilo, y *Pylint*, para el análisis estático del código.

El procedimiento de análisis de estilo, se implementa en la clase **Pep8Analy**, es una clase con los atributos necesarios para almacenar los errores de estilo, ficheros en que se producen, y estadísticas de los errores.

Pep8Analy recibe como configuración las rutas de los ficheros python a analizar, y los códigos de error que no deben ser tenidos en cuenta para este análisis. Y se inicializa invocando a la utilidad *pep8* como se indica abajo, para recopilar los errores detectados y un sumario de estadísticas.

```
pep8 --repeat --show-source {ignore=<codigos_error>
<ruta1_py ruta2_py...>
```

```
pep8 -q --statistics <ruta_py1 ruta_py2...>
```

Por otra parte, también se implementa una clase **PylintAnaly**, que con un funcionamiento semejante al de *Pep8Analy*, extrae y guarda de forma estructurada los errores *Pylint* y la nota de calidad del código obtenidas de la utilidad llamada de la siguiente manera.

```
pylint {disable=<codigos_error> <ruta_py1 ruta_py2 ...>
```

4.2.4. MÓDULO DE PRUEBAS DE EJECUCIÓN

```
TEST = (
    ('calc.py suma', 'python2 calc.py 1 suma 2', ('3',)),
    ('calc.py resta', 'python2 calc.py 2.0 resta 1', ('1',)),
    ('calc.py resta operando invalido', 'python2 calc.py dos resta 1',
     ("Error: Non numerical parameters",)),
    ('calcoo.py suma', 'python2 calcoo.py 1 suma 2', ('3',)),
    ('calcoo.py resta', 'python2 calcoo.py 2.0 resta 1', ('1',)),
    ('calcoo.py resta operando invalido', 'python2 calcoo.py dos resta 1',
     ("Error: Non numerical parameters",)),
    (
    'calcoohija.py mult', 'python2 calcoohija.py 2 multiplica 2.5', ('5',)),
    ('calcoohija.py div', 'python2 calcoohija.py 11 divide 4', ('2.75',)),
    ('calcoohija.py div por cero', 'python2 calcoohija.py 1 divide 0.0',
     ('Division by zero is not allowed',)),
    ('calcplus.py', 'python2 calcplus.py ' + fichpruebapath,
     ('15', '15', '15', '15', 'Division by zero is not allowed'))
```

Figura 4.2: Tuplas con la batería de pruebas para el módulo de pruebas

La clase **testPr** itera sobre cada una de las tuplas de prueba, ejecuta la prueba, e invoca a un pequeño módulo denominado **OutputAnaly**, que chequea a través de funciones de texto y de la librería *re* de Python que la salida recogida es compatible con la salida esperada, permitiendo cierta incertidumbre introducida por cambios en la capitalización, introducción de separadores o caracteres blancos, etc.

4.2.5. MÓDULO DE ANÁLISIS DE GIT

En esta versión de Misture, se implementa un análisis sencillo de repositorio Git. Dicho análisis, se realiza a través del parseo de la salida de la siguiente llamada a git.

```
git log <ruta_repo_alumno>
```

A la salida del *log*, se le extraen mediante funciones de la librería re de Python y los patrones adecuados los datos del autor del commit.

Asimismo, podemos proporcionarle un listado de palabras o raíces de palabras que consideramos clave en los *commits* de la actividad que corregimos por su significancia, obteniendo del análisis el número de apariciones, y en consecuencia, una medida sobre si el alumno etiqueta los *commits* con buen criterio.

```
DMONTH = {}
MONTH = ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
         'Oct', 'Nov', 'Dic')
numonths = tuple(range(1, len(MONTH) + 1))
for i in range(len(MONTH)):
    DMONTH[MONTH[i]] = numonths[i]


# PATTERNS.
GITLOG_PATTERN = ''.join([
    r"^commit\s+(?P<shacode>\S+)$\s",  # patrones ideados para flag re.M puesto
    r"^Author:\s+(?P<Author>.+)$\s",
    r"^Date:\s+(?P<Datetime>.+)$\s",
    r"^\s+(?P<Description>.+)$\s+"
])
DATETIME_PATTERN = r"^\w{3} (?P<month>\w{3}) (?P<day>\d{1,2}) (?P<hour>\d{2}):" + \
                   r"(?P<min>\d{2}):(?P<sec>\d{2}) (?P<year>\d{4})"
DESC_PATTERN = '|'.join([
    r"(calc)",
    r"(sum)|(rest)|(mult)|(div)",
    r"(commit)|(primer)|(DEFINITIVA)",
    r"(objeto)|(orientado)|(object)|(\s+o.o.\s+)",
    r"(herenc)|(herit)|(hij)",
    r"(csv)|(fich)|(comma.+sepparated.+value.+)"
])  # encouraged flags re.I Ignored Case
```

Figura 4.3: Regex utilizados para el análisis del log

4.2.6. MÓDULO DE COMUNICACIÓN

Con el objetivo de dotar de nuevas funcionalidades a Misture y automatizando y reduciendo el tiempo necesario para todas las tareas. Se implemento una funcionalidad con el objetivo de poder comunicar masivamente el resultado de las prácticas. Esta funcionalidad la implementamos en el módulo **mailing**.

En dicho módulo, a través la funcionalidad de las librerías *smtplib* y *email* de Python, o la API del servicio de emailing *SendGrid*, hay implementaciones para realizar el envío automatizado de los reportes, siempre que se proporcionen las credenciales de acceso.

4.3. DISEÑO DE BBDD

En la versión más reciente de Misture, se optó por emplear el sistema de base de datos MongoDB, con bases de datos no relacionales, sin esquema, o mejor dicho, de esquema dinámico.

La elección de MongoDB para almacenar todos los datos obtenidos de los análisis de los repositorios, pruebas, y comprobaciones de código, en principio facilita y nos permite ser más ágiles durante el desarrollo y prueba de los diferentes módulos y funcionalidades. Por una parte por no trabajar con un esquema rígido y por otra porque MongoDB encaja muy bien con los tipos de datos nativos de Python – objetos, listas, diccionarios-.

Adicionalmente, buscamos una librería ODM -Object Document Mapper- para MongoDB en Python, permitiéndonos operar con la BBDD como si fuera un objeto, facilitando aún más nuestra labor. En un primer momento se eligió **mongoengine**, aunque al final en Misture hemos utilizado **pymodm**.

Se debe recordar, que la estructura y validaciones de los campos definidos en las clases documento del ODM que representan los documentos de las colecciones de MongoDB son transparentes al propio MongoDB, se producen en el lado de la aplicación a través del ODM.

A continuación, vamos a enumerar los diferentes documentos considerados para poder llevar a cabo la funcionalidad básica de Misture.

[FIGURA XX: Representación de BBDD incluida en Misture]

LogError es la clase del documento donde se vuelcan los errores controlados de la ejecución de Misture.

- descripcion
- accion
- traza
- fecha
- actividad: referencia a **Actividad**
- correccion: referencia a **Correccion**

UsuarioMisture representa a un usuario del sistema Misture.

- login
- email
- rol
- date

Corrector es la clase heredera de **UsuarioMisture** que modela las particularidades del usuario corrector.

- actividades: referencia a **Actividad**
- rol
- superrol

Desarrollador es la subclasde de **UsuarioMisture** que representa al desarrollador.

- emails
- rol

DirectoriosActividad representan las rutas particulares donde se ubicarían los ficheros de la actividad a corregir.

- pbase
- pentrega
- presultados
- perrores
- ptest
- ppruebas

Actividad es la clase que representa una actividad, con sus datos básicos para ser corregida.

- descripcion
- corrector -¿referencia a **Corrector**
- desarrolladores -¿referencia a **Desarrollador**
- entregas -¿documento Entrega
- directorios -¿documento **DirectoriosActividad**
- ficheros_entrega

Correccion es la clase que representa a la ejecución de una corrección de una entrega de un alumno.

- fecha
- actividad -¿referencia a **Actividad**
- desarrollador -¿referencia a **Desarrollador**
- desarrolladores -¿referencia a **Desarrollador**
- gitanaly -¿documento **GitAnaly**

Entrega es la clase que modela los elementos necesarios para emprender la corrección de la actividad de un alumno particular.

- actividad: referencia a **Actividad**
- desarrollador: referencia a **Desarrollador**
- modo
- ubicacion
- correccion: referencia al documento corrección.

Prueba modela una prueba de caja negra.

- correccion: referencia a **Correccion**

- llamadas_ord
- outputs

Udfile representa a un fichero o directorio de una entrega.

- correccion -¿referencia a **Correccion**
- pathrel
- nombre
- tipo
- hijos -¿referencia a **Udfile**
- lenguaje
- sloc

Udfuente representa a un elemento de código dentro de un fichero fuente.

- udfile: referencia a Udfile
- parent: referencia a Udfuente
- nombre
- ubicacion: documento Udfuentecoor

Udfuentecoor es la clase que modela la ubicación del elemento de código dentro de una fuente.

- filaini
- filafin
- col
- pos

GitUser: usuario committer o author de Git.

- name
- email

GitBranch: datos de una rama de un repositorio Git de una corrección concreta.

- correccion -¿referencia a **Correccion**
- nombre
- nombre_remote
- targetid

GitCommit: representa un commit

- correccion -¿referencia a **Correccion**
- idhex
- orden
- author -¿referencia a **GitUser**
- committer -¿referencia a **GitUser**
- time
- offset
- mensaje
- palabrasclave
- commits_ant -¿referencia a **GitCommit**
- ficheros

GitAnaly es la clase que contienen los datos de analisis de un repositorio Git.

- commitentrega -¿referencia a **GitCommit**
- commitinicial -¿referencia a **GitCommit**

- numcommits
- palabras
- intervalos
- gitusers -¿referencia a **GitUser**
- gitbranches -¿referencia a **GitBranch**

CuestionExamen representa las preguntas del examen

- correccion -¿referencia a **Corrección**
- tipo_pregunta
- contenido_pregunta
- tipo_respuesta
- opciones_respuesta
- opciones_validas
- respondida
- respuesta
- fecha_respuesta

Capítulo 5

Resultados

Con la aplicación del proyecto Misture, pretendíamos, entre otros items a cumplir:

- Establecer un proyecto Python, que recogiera las funciones básicas, detectadas en las muestras de las actividades entregables de las asignaturas de programación de aplicaciones y servicios Web y multimedia en redes, adaptada a las pequeñas prácticas de los alumnos.

- Automatizar la funcionalidad básica detectada, minimizando la cantidad de pasos o funcionalidades semiautomáticas o manuales llevadas a cabo por el profesor en las experiencias piloto.

- Permitir cierta tolerancia -no permisividad- en la automatización a errores por parte de los alumnos en los códigos o en los ficheros entregables. Dado que estamos ante alumnos en proceso de aprendizaje. Y que la detección y rectificación automatizada de estos redundaría en reducir las intervenciones manuales y re-ejecuciones de los *scripts* de los diferentes pasos de la corrección.

- En consecuencia a los anteriores, disminuir el tiempo dedicado por el docente en el desarrollo o adaptación del código para la corrección de nuevas actividades, su ejecución, y la

recogida de resultados y reporte de los puntos críticos a revisar.

- Mejora en la generación, riqueza, y comunicación del *reporting* de los resultados de los diferentes análisis y correcciones que se ejecutan sobre los repositorios.

- Mejorar la recogida de información extraída en las correcciones, de forma persistente y debidamente relacionada en sistemas de BBDD, permitiendo nuevos análisis posteriores, y ahorrando tiempo y re-ejecuciones en caso de errores que requieran alguna intervención manual por parte del profesor.

- Interrelacionar nuestro sistema con una interfaz web, en una primera aproximación para la realización de exámenes de verificación de autoría y conocimiento del código del alumno. Asimismo, mejorar la experiencia piloto de forma que se pueda extender la funcionalidad de la interfaz web a mostrar.

De este compendio de deseos que inicialmente esperábamos cumplir, en el estadio en el que el proyecto Misture ha sido entregado, hemos llegado a los siguientes resultados.

En cuanto a establecer un diseño que recoja la funcionalidad básica detectada en las experiencias piloto, podemos estimar que se ha podido realizar, con el diseño descrito en el apartado precedente.

Asimismo, hemos construido una pequeña arquitectura, que en gran medida puede ser ampliada con nuevas funcionalidades y mejoras a partir de la propia estructura del proyecto y las BBDD propuestas.

Con respecto a los hitos de automatización, en algunas tareas concretas como el *reporting* y comunicación de los resultados, que requerían mayor intervención manual, ha sido posible introducir una mayor ganancia en tiempo y en reducción de las intervenciones manuales por parte del profesor.

Sin embargo, en otras comprobaciones, como la comprobación de los ficheros entregados, el estilo y errores de los códigos fuentes, pruebas, etc. La ganancia principal, más que en tiempo bruto, se produce en el hecho de disponer de los datos de los análisis almacenados sistemáticamente con una mayor riqueza e interrelacionados en las BBDD.

En cambio, en permitir cierta tolerancia en los procesos automáticos frente a pequeños errores de los alumnos, que son los que producen que tengan que realizarse correcciones manuales, para seguir pudiendo efectuar el resto de chequeos automáticos a esa práctica del alumno. Ha sido uno de los hitos más complicados de cumplir.

Se ha introducido alguna pequeña mejora, como la corrección con ayuda del algoritmo de Levenshtein de casos muy simples de equivocación de los nombres de los ficheros esperados, cuando se indica una lista obligatoria de ellos.

En cambio, no ha sido posible encontrar soluciones abordables en el ámbito de este proyecto y en un plazo razonable para automatizar la rectificación de otros pequeños despistes que se suelen producir en ocasiones por parte de los alumnos.

Por ejemplo, casos en los que no se definió estrictamente en el enunciado de la actividad propuesta, o el alumno cometió errores, con respecto a la forma de usar o leer los parámetros de los *scripts*, y esto pueden dificultar tareas como la ejecución de pruebas de caja negra, y exigen al tutor pasar a operar manualmente, buscar y rectificar el problema, y realizar re-ejecuciones, o no poder evaluar por completo al alumno.

En general, podemos decir que con respecto a las experiencias piloto se ha mejorado en varios aspectos las tareas de corrección automática. También que con Misture se sienta una posible base sobre la que ampliar y mejorar funcionalidades y el reporte de información extraída y sus posibilidades de análisis.

Sin embargo, nos hemos encontrado con que no es tan fácil de alcanzar un grado elevado de generalización de casos a abarcar, y de automatización plena y desatendida de la herramienta,

entre los pasos de configuración y de lectura de informes y reporte. Por tanto, no hemos podido abarcar todas las automatizaciones que nos hubiera gustado.

Capítulo 6

Conclusiones

6.1. LECCIONES APRENDIDAS

Una de las principales lecciones aprendidas, consiste en entender la complejidad que resulta del diseño de herramientas que permitan la automatización de tareas.

La posibilidad de generar estos automatismos en casos lo más generalizables posibles debe estar muy bien estudiada en sus elementos o circunstancias para evitar encontrarse ante un caso complicado de manejar y de controlar en sus condiciones más probables.

Sino, o no es una tarea netamente automatizable, o debe ser considerada inclusive en subtareas más manejables y cuyos elementos característicos sean lo más homogéneos entre sí.

También puedo considerar que he aprendido algo mejor a evaluar mejor que proyectos o APIs debo elegir, y adquirido la noción de lo complejo que puede llegar a ser mantener en el tiempo pequeños proyectos de código abierto por parte de pequeños grupos de desarrolladores, a menudo de forma altruista en su tiempo libre.

El mayor ejemplo de esto que digo es el cambio que tuve que realizar del ODM mongoengine por pymodm, el primero bastante potente y con bastante historial detrás y que potencialmente podía facilitarme mucho una integración con Django.

Sin embargo, con el paso del tiempo y al utilizar más funcionalidades de este ODM y algunos fallos heredados de diseño de este, me empezó a generar diversos problemas y comportamientos inesperados para los que no encontraba solución, teniendo que cambiar y llevándome a sufrir una pérdida de tiempo considerable.

No obstante, otra opción es elegir uno de estos proyectos que a uno le resulte atractivo y sus colaboradores estén abiertos a colaboraciones, y colaborar con ellos.

Otra importante lección aprendida por parte mía, es quizás reconocer mi exceso de perfeccionismo en la realización de este tipo de tareas, en ocasiones por encima de la capacidad técnica que pueda tener en el momento dado, o la disponibilidad de tiempo o recursos para llevar a cabo su ejecución.

En muchas ocasiones, no palpar adecuadamente estos límites junto al ansia perfeccionista, me ha llevado a callejones sin salida, códigos complejos que he tenido que desechar, y derivas de tiempo.

No obstante, por otra parte, esa característica propia de mí, aplicada al mundillo profesional, me ha llevado a ser muy bien valorado como analista en varios de los proyectos en los que he trabajado. Sin embargo, la cantidad de estrés que en ocasiones genera esto, es excesivo.

6.2. CONOCIMIENTOS APLICADOS

De una u otra forma, en la consecución de este proyecto, se han aplicado conocimientos adquiridos a lo largo de mi etapa universitaria cursando Ingeniería de Telecomunicación e Ingeniería Técnica en Informática de Sistemas.

Por una parte, hubo que recordar o aplicar conocimientos de la propia asignatura IARO, ya que alguna de las muestras de actividades empleadas para probar las funcionalidades correspondía a prácticas de contenido semejante a IARO.

Asimismo, nos aprovechamos del conocimiento adquirido en BBDD en las asignaturas de dicha temática. También de su uso en un framework como Django, cuya destreza adquirimos en materias como Sistemas y Aplicaciones Telemáticas.

También nos han sido útiles los conocimientos de otras materias como Ingeniería del Software y de Programación Orientada a Objetos.

No obstante, una vez afanados en encontrar relaciones, podríamos encontrarlas en todas las materias asociadas a la programación, BBDD, al desarrollo de software o a sistemas y aplicaciones web o de multimedia.

6.3. CONOCIMIENTOS ADQUIRIDOS

Durante la investigación y desarrollo de Misture, he entrado en contacto con un buen número de conceptos, utilidades y tecnologías, algunas de las cuáles incluso excedían con creces la capacidad y tiempo que deben otorgarse a un Proyecto de Fin de Carrera y no he llegado a aplicar en el proyecto y por tanto mencionar aquí.

Entre aquellos conocimientos adquiridos y practicados se encuentran:

- El DBMS MongoDB. Además de los conceptos relativos a bases de datos NO-SQL o no relacionales.

- Conocimientos sobre la herramienta Git y el manejo de repositorios distribuidos. Además del uso de los servicios Github y Gitlab.

- Aunque Python ya era conocido para mí, se puede decir que también he aprendido acerca de Python 3, sus diferencias con respecto la versión 2, y las características nuevas que trae.

- También se han adquirido conocimientos del uso y funcionamiento de diferentes herramientas de análisis estático, dinámico y de estilo del código, especialmente de aquellas orientadas al lenguaje Python: pep8, pylint, pymetrics, flake8, pyflakes, etc.

6.4. MEJORAS O TRABAJOS FUTUROS

En este subapartado, se describen las posibles mejoras o líneas futuras que permitan un mayor desarrollo del proyecto obteniendo mejoras en el mismo.

Dado que personalmente observé y he observado en el tiempo una gran cantidad de opciones donde poder profundizar, pero que por complejidad o falta de tiempo no ha sido posible efectuar. Voy a detallar algunas funcionalidades o mejoras posibles que se pueden aplicar al proyecto propuesto:

- Ampliar el uso de la interfaz web como cliente integral para la fase inicial de configuración de la corrección de una actividad.

- Ampliar las funcionalidades para hacer posible análisis entre de las diferentes actividades que con el tiempo se hayan corregido y pasen a formar parte de nuestro propio repositorio de correcciones.

- Emplear las capacidades de módulos o librerías como ast o astor para posibilitar funcionalidades más potentes en los análisis de código fuente y su estructura, o para realizar pequeñas correcciones asistidas del código fuente sobre pequeños errores frecuentes de los alumnos que pueden dificultar la ejecución de tareas como las pruebas de ejecución.

- Añadir nuevas formas de comunicación a la utilidad. Por ejemplo, añadir comunicación vía redes sociales tales como Twitter, Facebook o LinkedIn.

- Integrar capacidades propias para detección de plagio.

- Otra posibilidad a explorar, debido a la existencia de *scripts* cliente y de librerías para la visualización y análisis de los resultados del análisis, es integrar la funcionalidad de

MOSS en el propio Misture.

Apéndice A

Instalación y Uso

Con el sistema Misture propuesto, si se desea efectuar una corrección con una parte o la totalidad de las funcionalidades básicas ya implementadas, sería necesario:

1. El directorio del profesor donde deposita los ficheros auxiliares a la corrección, tales como:

   - Fichero con el listado de logines únicos de alumnos, emails, y ruta o url del repositorio individual. En este caso obligatorio.

   - Fichero con las preguntas estáticas de la prueba o examen que se mezclaran con las generadas de autoría.

   - Ficheros auxiliares para la ejecución de pruebas sobre el código: códigos de un servidor o cliente de prueba para lanzarlo contra el del alumno, xmls de configuración, ficheros de entrada sobre el que lanzar una prueba, etc.

2. Una copia del *script* principal, que lee las configuraciones e invoca los diferentes submódulos de corrección.

3. Se utiliza a modo de plantilla, teniendo que ajustar la configuración y código del *script*.

4. Estos ajustes a los módulos que realmente se van a utilizar entre aquellos que son opcionales, que llamadas se van a ejecutar en las pruebas y que resultado u listado de resultados válidos se pueden esperar.

5. Asimismo, se pueden ajustar otras constantes como el listado de errores o tipo de errores de pep8, pylint o hackings ignorar en un análisis de estilo y código.

6. También, en caso de activar la comprobación de xml o pdml, se indicarían cuáles comprobaciones, entre las implementadas, se deberían realizar, como la cuenta de trazas de determinado tipo en la captura Wireshark, la presencia de cierto atributo o valor en una etiqueta concreta, etc.

Apéndice B

Requisitos

Para la ejecución del código de Misture.

En cuanto a librerías Python, si no se indica lo contrario, su instalación se aconseja realizar el gestor de paquetes Python Pip para instalar los paquetes necesarios en la versión 3 de Python, de la siguiente manera.

```
pip3 install <nombre_paquete>
```

Dada la cantidad de paquetes y dependencias manejados, se detallarán las dependencias principales, siendo instaladas todas las dependencias en las versiones disponibles o las requeridas por el propio paquete.

Además, por la misma razón, es una buena práctica configurar un "Virtual Environment" de Python específico para la configuración de todo el entorno necesario y la ejecución de este proyecto.

Una vez aclarado todo esto, definimos los principales requisitos:

- Interprete de Python: durante el desarrollo, se ha empleado principalmente el interprete de la versión 3.4, aconsejándose la instalación de la versión 3.4 o superior.

- Asimismo, se dispuso de un interprete de la versión 2, en concreto de la versión 2.7, para el testing sobre códigos de Python 2. Aconsejándose, en caso de la corrección de prácticas en esa versión de Python, de un virtualenv con la versión más reciente utilizada por los alumnos.

- Sistema gestor de bases de datos MongoDB: se ha empleado la versión 3.2.13.

- La librería ODM "pymodm" de python: se aconseja usar la versión 0.4.0 o superior, teniendo en cuenta en la documentación de esta librería, se indica que está testeado para la versión de MongoDB que se tiene instalada.

- Sistema de control de versiones Git: se recomienda tener instalada la versión 2 de este herramienta, ya que esta documentado y asimismo hemos detectado en diferentes pruebas con la librería que algunos de los comandos y opciones de Git han modificado su comportamiento, sus opciones por defecto, o el formato de su salida, afectando por tanto al comportamiento de pygit2.

- Librería pygit2. El sistema Misture está probado con la versión 0.25.0

- Librería Python de la herramienta flake8: se ha empleado la versión reciente 3.3.0.