ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
DE TELECOMUNICACIÓN

INGENIERÍA DE TELECOMUNICACIÓN -
INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

PROYECTO FIN DE CARRERA

MiStuRe: Minado de repositorios de estudiantes orientado al análisis del aprendizaje.

Autor : Carlos González Sesmero

Tutor : Dr. Gregorio Robles

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 Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de Julio de 2017, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de Julio de 2017

"[...] 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, el profesorado tiene una doble exigencia. Por un lado, proporcionar una docencia con un método de evaluación continua y muy dinámica. Por otro, la exigencia de una intensa actividad investigadora.

En este escenario, una materia tiene como pretensión unos objetivos generales, unos objetivos específicos, y dotar de un conjunto de habilidades al alumno. Resulta positiva cualquier mejora o sistema que refuerce al profesor, optimice el tiempo disponible para evaluar, o incremente la frecuencia y cantidad de información que realimenta a alumno y profesor respecto al estado de estos objetivos.

Por otra parte, nos encontramos con disciplinas como la programación, donde es posible el tratamiento automático, la obtención de patrones o de mediciones de los entregables donde los alumnos demuestran su destreza y su evolución. En éstas, puede ser útil la aplicación de métodos o el uso de sistemas que permitan extraer sistemáticamente información valiosa con más rapidez que una metodología tradicional. La utilidad está en permitir de una forma más dinámica y objetiva la transmisión y el refuerzo de conocimientos, la detección de los errores y de los hábitos mejorables a nivel individual y grupal.

Tomando como referencia unas experiencias previas en unas asignaturas en las que participé, donde se emplearon estos conceptos, se va a intentar construir una propuesta de un sistema más automatizado y generalizable. Este sistema estará orientado a alumnos con un nivel básico de programación, en el que aun están formando sus destrezas y hábitos, y será aplicado a materias donde la programación es un soporte para el aprendizaje de otros conceptos como los relativos a redes y servicios web.

Eminentemente, estaremos en un ecosistema donde los alumnos manejarán el lenguaje Python, caracterizado por su simplicidad y alta interoperabilidad, y la herramienta colaborativa Git como elementos comunes.

Summary

This Final Project, named MiStuRe, being inspired by past educational experiences, develops an automatic tool both for the programming exercises correction as for the generation of simple exams.

The goal of MiSture concerned to professor, is decrease the time spend about corrections and give him a lot of new information about programming capabilities and subject's knowledgement among students to improve the academic progress of class.

Also, the purpose is that students gain a faster and richer feedback than handmaked correction about their delivered tasks, and maybe obtain plus motivation about the subject or in order to be better.

Índice general

Índice de figuras

Abreviaciones

API	—	Application Programming Interface.
BBDD	—	Bases de datos.
DBMS	—	Data Base Management System.
JSON	—	JavaScript Object Notation.
MOSS	—	Measure Of Software Similarity
MSR	—	Mining Software Repositories.
NoSQL	—	Not Only SQL.
ODM	—	Object-Document Mapper.
PEP	—	Python Enhancement Proposal.
REGEX	—	REGular EXpression.
SAX	—	Simple API for XML.
SMIL	—	Synchronized Multimedia Integration Language.
SQL	—	Structured Query Language.
XML	—	Extensible Markup Language.

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, el acrónimo de MIning STUdent REpositories, que guarda semejanza gráfica o sonora con los términos español mistura e inglés mixture.

Misture representa 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 se engloba dentro de un ámbito educativo universitario en el que se desea poder realizar por medio de *scripting* tareas de forma rápida y automática, 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 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 que llevó a la consecución de MiStuRe no es propia, sino que parte de una propuesta del tutor, personalmente no me resultó una propuesta carente de motivación, 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, y 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, surgió el impulso vital para 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 que pueden permitir la implementación de un escenario así en cuanto a temas y tecnologías con los que interactuar.

1.3. Estructura de la memoria

Esta memoria, relativa al Proyecto Fin de Carrera denominado como MiStuRe, al cual 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 esta sección, 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 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 sucesivas secciones 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, sus componentes, que elementos y utilidades nos sirven en su implementación y la estructura de las BBDD utilizadas.

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

La cuenta de capítulos queda cerrada con aquel denominado "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 particularidades de las dependencias necesarias para la ejecución del proyecto, y la bibliografía con 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 se pueda trasladar 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, y produciéndose la menor cantidad posible de errores imprevistos, que aumenten el tiempo empleado en efectuar procesos manuales.

Es deseable que estas correcciones puedan realizarse lo más desatendida y automáticamente posible, permitiendo liberar tiempo al profesor y disminuir el tiempo de respuestas, además de proporcionarles 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 con poco retraso con respecto al día de entrega.

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 objetivamente su evolución desarrollando esa actividad El directorio de trabajo debería contener 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 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 de la manipulación 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. 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 buena primera aproximación y permite abordar todo el proceso de corrección que de otra forma llevaría días o semanas para un único docente, no produciría tanta cantidad de información ni resultados susceptibles de evaluar, y por tanto se pierde capacidad de incidir sobre estos durante las siguientes sesiones.

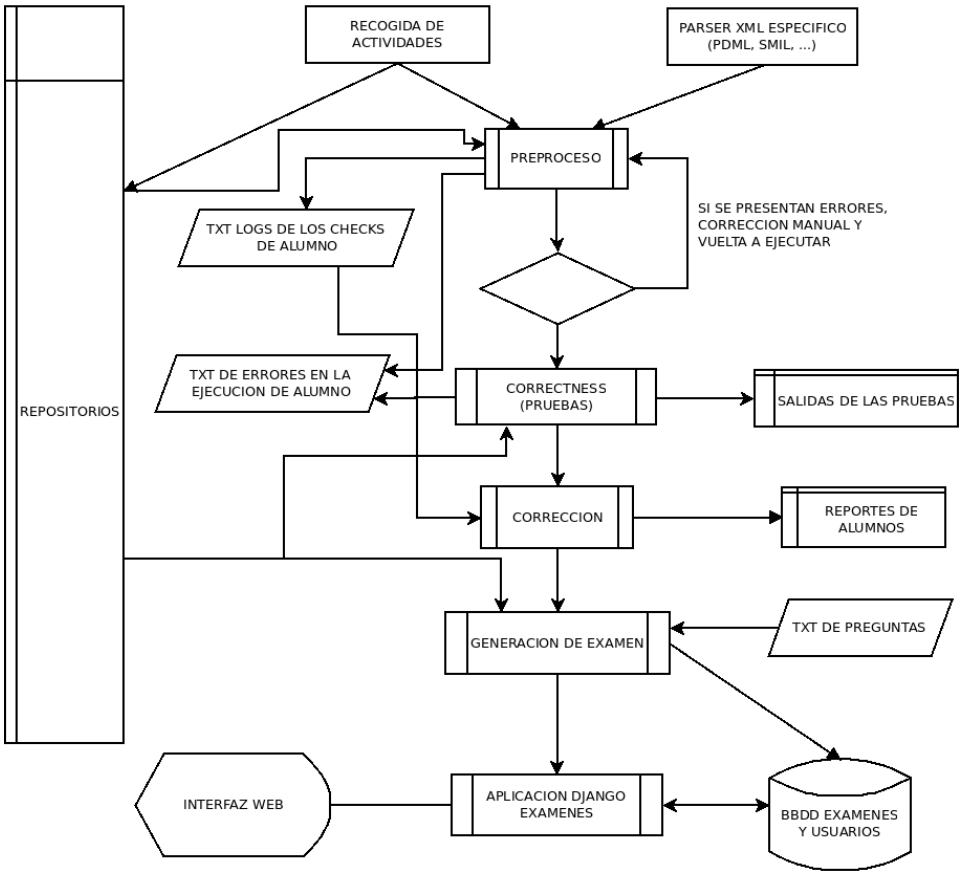


Figura 2.1: Flujo de los scripts semiautomáticos de la experiencia piloto

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 mucha 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

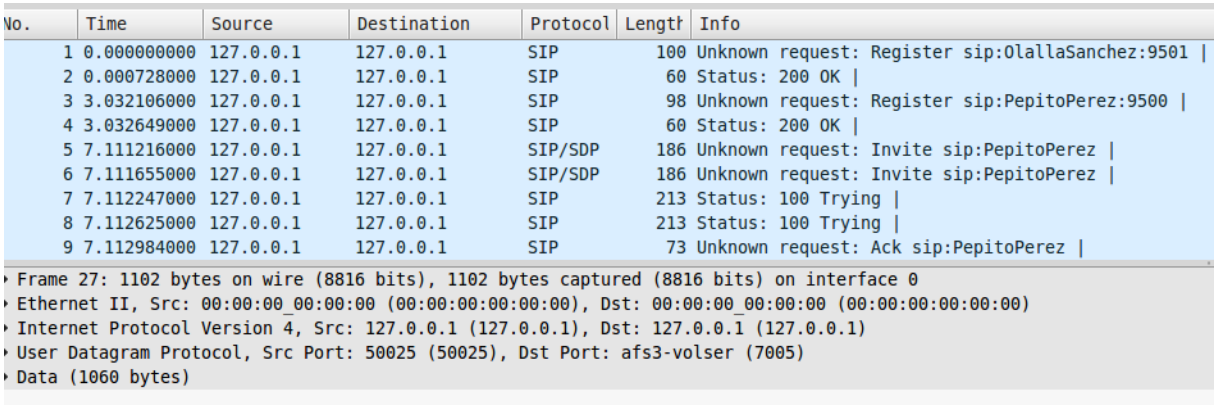
En este módulo se lee y establece la configuración inicial con las especificaciones de la entrega tales como los usuarios de los alumnos, localización de repositorios, listado de ficheros a entregar, etc.

Una vez establecida la configuración. Se lanzan diferentes comprobaciones sobre los ficheros mediante funcionalidad Python o invocando terceras utilidades mediante 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*. Estas comprobaciones son:

- 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 incluyen capturas de 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 lleva a cabo en este análisis es implementar un pequeño parser SAX en Python específico para la actividad a corregir. Este parser realizaría las comprobaciones sobre las etiquetas, atributos o los valores de estos en el XML. Con el resultado de ejecutar este parser sobre el XML, se puede concluir si la traza o el XML tiene los contenidos o elementos esperados y si el alumno ha realizado su tarea con éxito.

```
  <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 el parser 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 contra resultado esperado por el profesor, en el caso que sean actividades más sencillas o que tengan definido un formato de salida muy estricto en el enunciado planteado para la actividad. En caso contrario, la salida se almacena dentro del *pool* de ficheros de reporte del alumno para que sea inspeccionado por el profesor.

2.2.5. Posproceso

La funcionalidad de este *script* es parsear los logs del preproceso con el resultado de los *checks*. Así se genera un reporte en formato texto con el detalle sobre las comprobaciones realizadas y su resultado, el desglose de sus errores de estilo y avisos sobre líneas de código que incurren en alguna práctica no considerada de calidad. Según el caso, también se reporta 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 evalúen.

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 código 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 la salida que produce un código o hacer un cambio sobre él.

```
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 el tiempo necesario son publicados en su servidor web.

MOSS mide la similitud de entre ficheros de texto a través de métodos estadísticos. Según el tipo de ficheros y su contenido, es habitual encontrarse ante falsos positivos, especialmente si cotejamos 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 se ha estudiado todo el escenario de nuestro problema y la funcionalidad semi-automática de los *scripts* de corrección, se definió 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 cláusulas “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– puede detectar ciertos hábitos o malas prácticas desarrollando o aplicando los conceptos teóricos de la asignatura, 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 de texto, 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. También se dispone del reporte de incidencias u observaciones realizadas durante la ejecución y realización de algunas de las correcciones, para intentar buscar mejoras en las funcionalidades que permitan reducir los puntos críticos en la corrección, que incapacitan la acción del *script* y obligan 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 recoja las funciones básicas, detectadas en las muestras de las actividades entregables de las asignaturas de programación de aplicaciones y servicios web y multimedia sobre 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.
- Analizar patrones de corrección idóneos. Como en nuestro contexto nos hallamos ante alumnos en proceso de aprendizaje de destrezas de programación. Se debe 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.

- 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.
- Mejorar 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 cual tenía un breve conocimiento básico – los comandos *status, log, add, commit, pull, push* – sólo empleados para proyectos 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 el estado del ámbito del proyecto y de las tecnologías utilizadas para la realización del mismo.

3.1. Mining Software Repositories

Este Proyecto de Fin de Carrera, puede asociarse al área de los Mining Software Repositories -MSR-. El objetivo de la comunidad MSR es aprovechar todos los datos que se generan en el proceso de desarrollo software.

En general, estos datos se hallan en repositorios de software, como por ejemplo repositorios de control de cambios del código fuente -como Git-, repositorios de seguimiento de bugs y errores, o repositorios de comunicaciones que almacenan las comunicaciones entre el equipo de desarrolladores como es el caso de las listas de distribución.

La comunidad MSR se dedica a encontrar fuentes de información provenientes de los proyectos software, estudiar sus características, encontrar técnicas para extraer de forma efectiva la información y el aprovechamiento de esos datos para mejorar los requisitos, la calidad y la trazabilidad de los proyectos, y estudiar el impacto de diferentes fenómenos en un desarrollo.

Gran parte del conocimiento y evolución en este campo, se presenta anualmente en la conferencia de MSR, cuya décimo cuarta edición se ha celebrado este año en Buenos Aires, conjuntamente a la Conferencia Internacional de Ingeniería de Software.

En esta conferencia, se presentan los artículos sobre MSR que han sido aceptados tras su revisión y entre las actividades se encuentra la propuesta de un desafío sobre un conjunto de datos propuesto, para el cual los que aceptan ser desafiados tienen que solucionar a través de sus herramientas o sistemas MSR y presentar un reporte de los resultados.

Las temáticas dentro de los MSR, entre muchas otras, podemos enumerar, por ejemplo:

- Predicción de las características del software mediante el análisis del repositorio software.
- Caracterización, clasificación y predicción de defectos en el software mediante el análisis de los repositorios software.
- Análisis de cambios o de nuevas tendencias.
- Técnicas de visualización y modelos para los conjuntos de datos minados.
- Privacidad y ética en los MSR.
- Minado sobre cualquier elemento asociado a un proceso de desarrollo software. Los ya mencionados, pero también otros como licencias, *logs* o las opiniones en los centros de aplicaciones y sitios de opiniones sobre los programas.

Centrándose más en la parcela relativa a este proyecto. Por otra parte, las experiencias piloto que suponen el punto de partida de este Proyecto de Fin de Carrera, fueron analizadas y expuestas en *Gregorio Robles y Jesús M. González-Barahona (2013). Mining student repositories to gain learning analytics*. En éste, se expone el proceso de corrección, que ya describimos en el capítulo 2.

Entre las conclusiones del articulo, se expone que proporciona importantes beneficios al facilitar la evaluación continua y realimentación frecuente a los alumnos. El otro hecho que se concluye, es que un sistema de estas características, que se ha tenido que refinar en un proceso

iterativo, no es posible hacerse completamente automático, y se deben tomar medidas para minimizar las tareas manuales del proceso, la principal, es que los estudiantes sigan estrictamente las instrucciones indicadas para la realización de la actividad.

Asimismo, a la hora de explorar formas y herramientas para explotar los datos brutos de las entregas de los alumnos, me resultó útil a nivel conceptual el análisis de las diferentes fuentes que nos podemos encontrar en un proyecto de software, que se realiza en el artículo *Robles, G., González-Barahona J. M., Izquierdo-Cortazar, D. y Herraiz, I. (2009). Tools for the Study of the Usual Data Sources found in Libre Software Projects*.

3.2. Tecnologías utilizadass

3.2.1. Python

Python es un lenguaje de programación interpretado, usa tipado dinámico y es multiplataforma. Además se trata de un lenguanje de programación multiparadigma, permitiendo programación orientada a objetos, imperativa y funcional. Además posee una extensa biblioteca estándar y existen infinidad de librerías de terceros.

Dadas sus características, Python es muy útil como lenguaje para scripting y desarrollo rápido de aplicaciones de todo tipo.

Esta licenciado bajo la *Python Software Foundation License* desde la versión 2.1, licencia declarada libre, y compatible con GPL.

Fue creado por Guido van Rossum -el Benevolente Dictador Vitalicio- en los Países Bajos a finales de los años 80 y su nombre proviene de los humoristas británicos Monty Python.

La versión 1.0 fue lanzada en 1994, la 2.0 en Octubre de 2000 y la 3.0 fue lanzada en Diciembre de 2008. Actualmente, coexisten dos versiones de Python, la 2 y la 3, con diferencias que las hacen incompatibles, siendo las últimas versiones la 3.6.1 y la 2.7.13.

Python pretende ser un lenguaje fácil de leer. Para delimitar bloques se emplea la indentación en blancos en lugar de las llaves y los puntos y coma. En el mundillo Python, existe

una filosofía con unos principios de Python en cuenta a legibilidad y transparencia, recogida en el documento "El Zen de Python" (PEP 20). El código que siga sus principios se denomina *pythonic*.

3.2.2. Django

Django es un framework de aplicaciones web open source, escrito en Python, que aparenta seguir el patrón de diseño Modelo-Vista-Controlador. Proporciona al usuario una serie de componentes habituales en las aplicaciones web, como autenticación, sesión, administración, formularios, gestión de archivos, ya preparados para ser usados.

Está orientado a realizar desarrollos rápidos, por lo que incluye un servidor web ligero para facilitar esto, aunque soporta *Apache2* para la etapa de producción y servidores *WSGI*. Las bases de datos soportadas son *PostgreSQL*, *MySQL* y *SQLite3*.

Django fue liberado al público bajo licencia BSD en 2005, siendo desde 2008 la *Django Software Foundation* quien mantiene el proyecto. La última versión es la 1.11, que será la última que soporte Python 2.

3.2.3. MongoDB

MongoDB es un sistema de base de datos NoSQL lanzado en 2009 y ampliamente adoptado por la industria. La última versión es la 3.4.4. Es orientado a documentos, los cuales se guardan en colecciones de documentos de una estructura similar a JSON -BSON-. La equivalencia entre elementos de una BBDD SQL y NoSQL, es la siguiente:

- Tabla - Colección
- Registro - Documento
- Columna - Campo

Hace uso de esquemas dinámicos y no existe un esquema predefinido, pudiéndose alterar en cualquier momento, e incluso tener una misma colección varios documentos con distintos campos. Es un paradigma muy válido y eficaz en aplicaciones que necesiten almacenar datos semi-estructurados y facilitan los procesos de desarrollo iterativo.

No es recomendable para aplicaciones que usen intensivamente *JOINS* o requieran transacciones en lugar del bloqueo a nivel de documento que realiza MongoDB. Asimismo, puede presentar problemas de consistencia entre documentos, las escrituras no son verificables y puede perderse información.

MongoDB viene por defecto con una consola construida sobre Javascript, realizándose las consultas y operaciones en este lenguaje, y pudiéndose usar la sintaxis elemental de este lenguaje.

Asimismo, MongoDB tiene drivers oficiales para los principales lenguajes de programación. Para Python, el driver recomendado es PyMongo.

PyMODM

PyMODM, es un ODM sobre el driver de MongoDB PyMongo, desarrollado por ingenieros de MongoDB. Es una librería que proporciona los mecanismos para mapear la estructura de la base de datos a objetos Python, pudiéndose usar la información de la base de datos como si se tratase de un objeto.

3.2.4. GIT

Git es un sistema de control de versiones de software libre, diseñado por Linus Torvalds y lanzado en 2005. Se caracteriza por ser una herramienta distribuida, muy eficiente de proyectos grandes y que soporta muy bien el desarrollo no lineal en un proyecto, con la gestión de ramas, diferencias y mezclado de versiones.

En Git, los datos se modelan y almacenan como una sucesión de instantáneas de un pequeño sistema de archivos cada una, salvo el caso de ficheros idénticos entre dos instantáneas, que se

referencian mediante un link. Además, todos los objetos en git son verificados mediante SHA-1 a bajo nivel, por lo que Git posee integridad.

Para operar con el repositorio o el historial, no es necesario estar conectado al servidor, salvo para sincronizar los cambios, pudiendo operar todo el tiempo en local.

PyGIT2

Librería externa de Python, que proporciona en Python la funcionalidad esencial de Git a través de *Libgit2*, la API en C multiplataforma.

3.2.5. Análisis de código fuente

El análisis del código fuente, es un tipo de análisis dentro del análisis estático de software, habitualmente realizado por herramientas automáticas, que evalúa el software sin ejecutarlo, para permitir mejorar el código, sin alterar la semántica.

En un análisis estático de código automático, se incluyen análisis de sintáxis del código fuente, y diferentes reglas a aplicar sobre determinadas estructuras de código.

En el caso del análisis humano, este se denomina comprensión de programas o también revisión de código, que es complementario al automático y permite evaluar aspectos que un sistema automático no suele cubrir, como la arquitectura, el diseño, o la forma de trabajar con elementos externos.

Los objetivos de estos análisis puede ser tales como la verificación de propiedades del software para seguridad, detección de código vulnerable, la mejora de la calidad, o mejorar el mantenimiento y desarrollo.

No obstante, el análisis de código fuente no es suficiente para evaluar o mejorar un desarrollo en todos sus aspectos, por lo que debería complementarse con otras técnicas de análisis de software, por ejemplo, los test.

UTILIDADES DE ANÁLISIS DE CÓDIGO FUENTE EN PYTHON

Para la realización de análisis de código en Python, nos hemos ayudado de varias herramientas, introducidas a continuación.

La utilidad Pep8, que recientemente se ha rebautizado como *Pycodestyle*, es un *checker* de la guía de estilo de Python. Esta utilidad, se ejecuta sobre ficheros fuente Python, y nos proporciona un listado con la localización, el código de error E*** o W*** y la descripción de los incumplimientos con respecto a la guía de PEP8.

Pylint, es otra utilidad de verificación de código Python bastante completa, que verifica estilo, bugs y calidad del código. También si los nombres de los elementos están bien formados con respecto a los estándares y si las interfaces declaradas son implementadas. Es una herramienta comúnmente integrada en los entornos de desarrollo y editores de código.

Flake8 es una utilidad que auna las comprobaciones de estilo de Pep8, de errores de lógica de Pyflakes, y de complejidad circular. Su diseño permite la extensión mediante la inclusión de extensiones. Por defecto incluye la extensión McCabe, que generaría errores con el codigo C9**, si detecta casos de complejidades mayores a un umbral dado. Las comprobaciones de errores de lógica de Pyflakes se etiquetan mediante códigos F***.

Otra extensión popular es Hacking, que introduce errores con el código H***, y sus comprobaciones son las contempladas en *OpenStack Style Guidelines*, basadas en *Google Python Style Guidelines*.

Existe una completa documentación de los códigos de error en los manuales de las diferentes utilidades mencionadas.

3.3. 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.

En otros casos, aunque Python es un lenguaje multiplataforma y Misture emplea muchas librerias y utilidades que también lo son, 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.

3.3.1. 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.

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

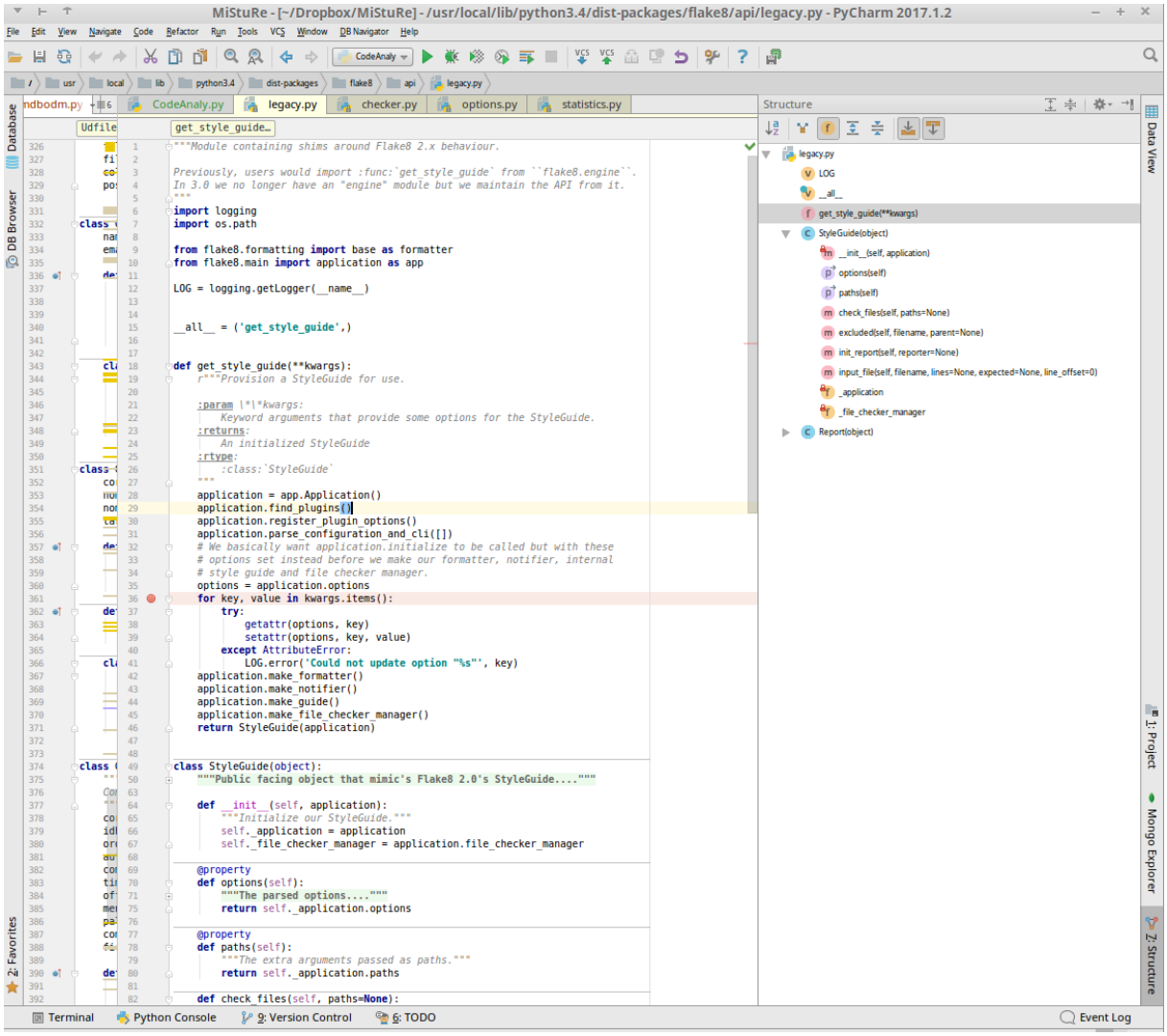


Figura 3.1: Interfaz de PyCharm

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.

3.3.2. Robomongo

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

Entre sus capacidades, integra la funcionalidad del cliente shell de MongoDB, permite la escritura de scripts y la vista de las propiedades, configuración y colecciones de las BBDD MongoDB del equipo. Resulta de gran ayuda navegación entre las diferentes colecciones y documentos y la posibilidad de visualizarlos en diferentes formatos como tabla, árbol, o la tradicional vista de texto en formato JSON.

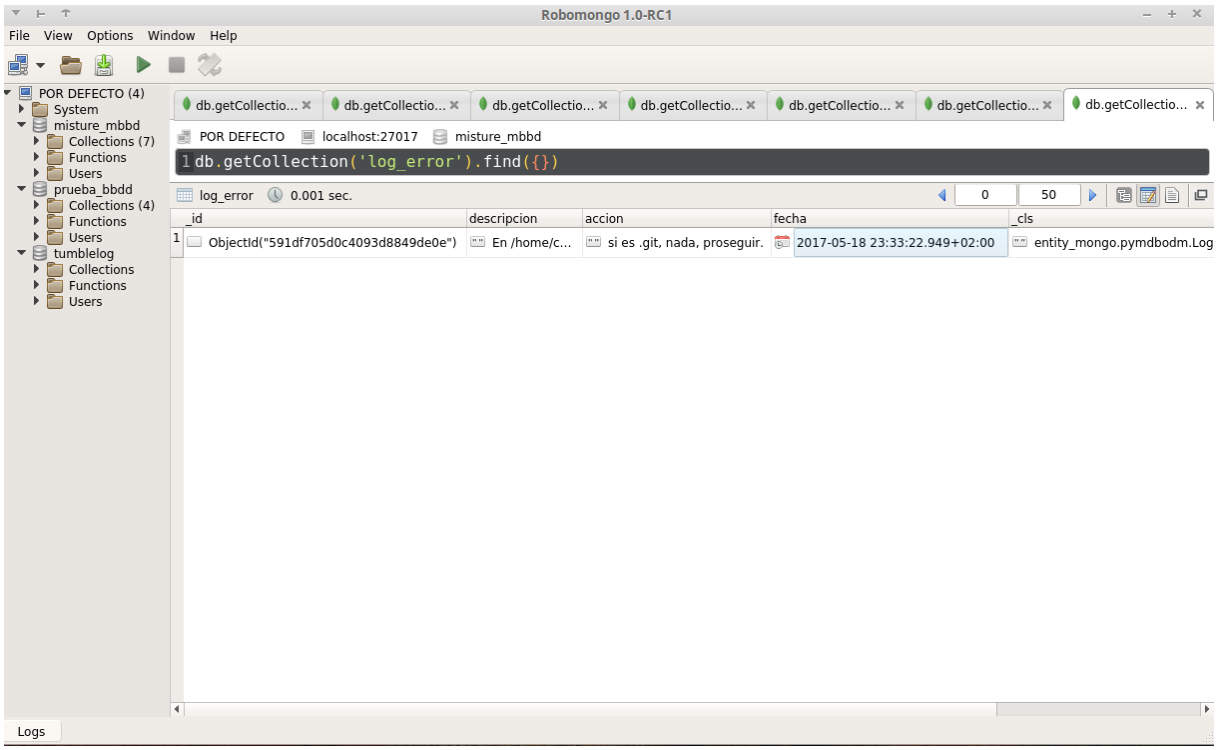


Figura 3.2: Interfaz de Robomongo

Robomongo era 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

Arquitectura y diseño.

En este capítulo, vamos a describir la metodología de desarrollo del proyecto Misture, y el diseño y arquitectura en varios momentos de su desarrollo, respondiendo a una funcionalidad creciente y más rica.

Por último, se detalla de forma lo más aproximada posible la estructura de la base de datos no relacional que se emplea en el estado final del proyecto.

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 que sea funcional, con el que se pueda interactuar y nos proporcione 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 afectan a la última interacción y no a todo el conjunto, volviendo al incremento previo en el peor caso.
- En el caso de este proyecto, permitía avanzar paso a paso en el desarrollo del sistema mientras se 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. Estas empezaban con prácticas sencillas o introductorias, a las que se le iban añadiendo elementos, funcionalidades y complejidad hasta llegar a la práctica final, que en realidad era 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 las 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 contar los errores e imprimirle al alumno qué errores comete, qué significan y en qué 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 dispone de las configuraciones básicas para obtener los repositorios y conocer los requisitos básicos como qué 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. Es decir, realizábamos un incremento para agregar funcionalidad básica de análisis de código Python, posteriormente otra iteración para la extracción de información básica del repositorio Git, etc.

Entre dos incrementos, las únicas entradas de las que disponemos son los requisitos y el resultado de la interacción entre el cliente y el resultado del incremento anterior. En este caso el cliente somos nosotros probando los repositorios de las actividades de los alumnos.

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. Iteración 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 en las transacciones con las utilidades externas y para generar logs de errores e imprimir el reporte final ordenado. También se utilizaron estructuras dinámicas como listas, 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 de clases, es una primera versión del modelo de datos que será usado en la iteración 2 y que se detallará posteriormente en el apartado DISEÑO DE BBDD.

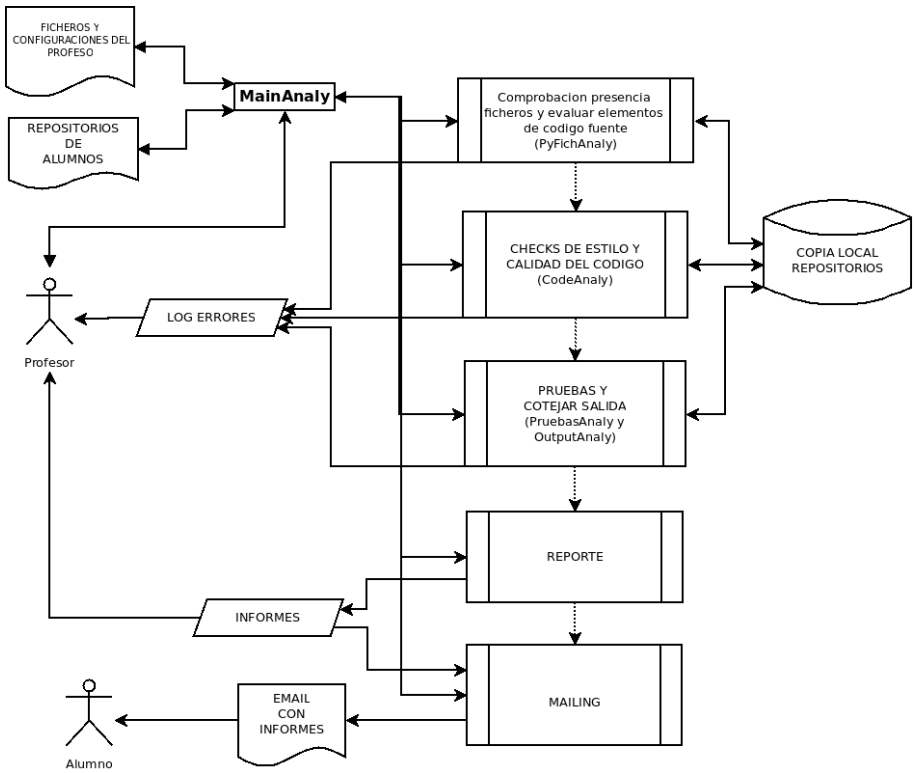


Figura 4.1: Flujo de la iteración 1.

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 salidas 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, inicializar los logs para la actividad, y establecer una ubicación para los reportes individualizados.

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

Al final del flujo se invoca a los objetos Misture para que se vuelque su información ordenada a los ficheros de reporte del alumno asociado, y se ordena al módulo de comunicación enviar en lote todos los reportes a los alumnos.

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 definida de ficheros que se esperan de los alumnos.

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. Este algoritmo tiene la capacidad de 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*"
INTLINE = 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 4.2: Patrones utilizados para identificar elementos de código y su nombre

4.2.3. Módulo de análisis de calidad de código

En el módulo *CodeAnaly*, 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=<cods_error> <rutas_py>
$ pep8 -q --statistics <rutas_py>
```

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

A través del módulo *PruebasAnaly*, en esta versión de Misture se ejecutan y almacenan los resultados para ser asociados a la corrección del alumno, una batería de pruebas que debe ser proporcionada al ser invocado la interfaz del test, implementado en la clase *testPr* del módulo.

Cada elemento de la batería de pruebas es una prueba simple a través de una llamada a uno de los programas o scripts del alumno, sobre la ruta de su repositorio. Cada prueba, esta representada por una 3-tupla de etiqueta, prueba a ejecutar, y un tercer elemento que es una tupla con el contenido de las líneas que debemos recoger a la salida estándar de la prueba para dar por válida la prueba.

```
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 4.3: Tuplas con la batería 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 4.4: 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 implementó 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 y cuando se proporcionen las credenciales de acceso.

4.3. Iteración 2:

Una vez descrito el diseño y funcionalidad de Misture en el estado anterior, podemos observar que el diseño propuesto en ese punto, introduce algunas pequeñas mejoras en cuanto a funcionalidades, organización y modularidad del sistema, especializando las tareas y diferentes tipos de análisis .

Asimismo, se ha establecido un universo de objetos que modela aproximadamente los elementos que nos vamos encontrando en una corrección automática y la generación de resultados de *checks*, y así disponer de toda la información extraída en los análisis de una forma estructurada.

De esta forma, se facilita la recuperación de la información, pudiéndose elaborar diferentes informes al final del proceso corrección partiendo de la misma estructura de información, en lugar de componer una combinación heterogénea y por separado para alumno y profesor de trazas de diferentes ficheros sobre la marcha. También se facilita la posibilidad de añadir con más facilidad en pasos posteriores del proceso, otros análisis a partir de la información estructurada de la corrección. Por ejemplo, para añadir un pequeño análisis de la actividad a nivel de grupo, mediante la agregación de la información sobre cualquiera de los análisis.

Aun así, a pesar del trabajo desempeñado, la funcionalidad no supone una mejora cualitativa con respecto al sistema de *scritps* de preproceso y corrección que se plantea en las experiencias piloto de los que partimos. Además, seguimos circunscritos a la corrección de entregas de tamaño reducido, con un pull de ficheros cerrado, y comprobaciones basadas en el parseo de la salida de trazas de terceras utilidades.

Por tanto, esta aproximación, una vez nos planteamos ampliar la funcionalidad, añadir más información para cada uno de los análisis, y trasladarlo a las correcciones de entregas más complejas y avanzadas, nos damos cuenta que es algo limitada para tales intenciones.

Tras recapitular estas ideas, es posible vislumbrar qué líneas vamos a seguir en la mejora del proceso. Por tanto, en los sucesivos incrementos de funcionalidades, se buscaron las siguientes mejoras:

- Estructurar la información en BBDD
- Especializar aún más los módulos de análisis.
- Búsqueda de APIs Python de las utilidades que se emplean para el análisis, o de alternativas que nos ayuden a tal fin.

A raíz de los cambios introducidos en el diseño de los sucesivos incrementos que realizamos en esta segunda versión de Misture, la estructura del código resultante se amplió, constando el proyecto con la siguiente estructura:

- El *script* o módulo principal, desde el que se deben especificar las configuraciones y entradas -listados, ruta de ficheros del profesor, etc- necesarias para la corrección, y se invocan las llamadas a los análisis disponibles con los parámetros que el profesor quiera lanzar.
- Un paquete de entidades –denominado *entities*- donde se representan a los objetos del universo Misture, mapeados como documentos a una BBDD MongoDB.
- Un paquete que contiene todos los módulos especializados que proporcionan acceso a cada pequeña funcionalidad de análisis implementada a través de, denominado *Analyx*.
- Se añade un paquete, cuya función consiste en hacer de intermediario entre los tres elementos anteriores, denominado *funcionalityx*. Proporciona al módulo principal la interfaz sobre las funcionalidades de análisis a usar. Después, para cada funcionalidad, se encarga de invocar las operaciones y análisis especializados necesarios de los módulos del paquete *analyx*. Y por último, es el encargado de interaccionar con las entidades del paquete *entities*, de forma que la información que recibe de los análisis la almacene estructurada en los objetos del ODM, que se almacenaran de forma persistente en la BBDD.

A continuación, se describen dichos paquetes.

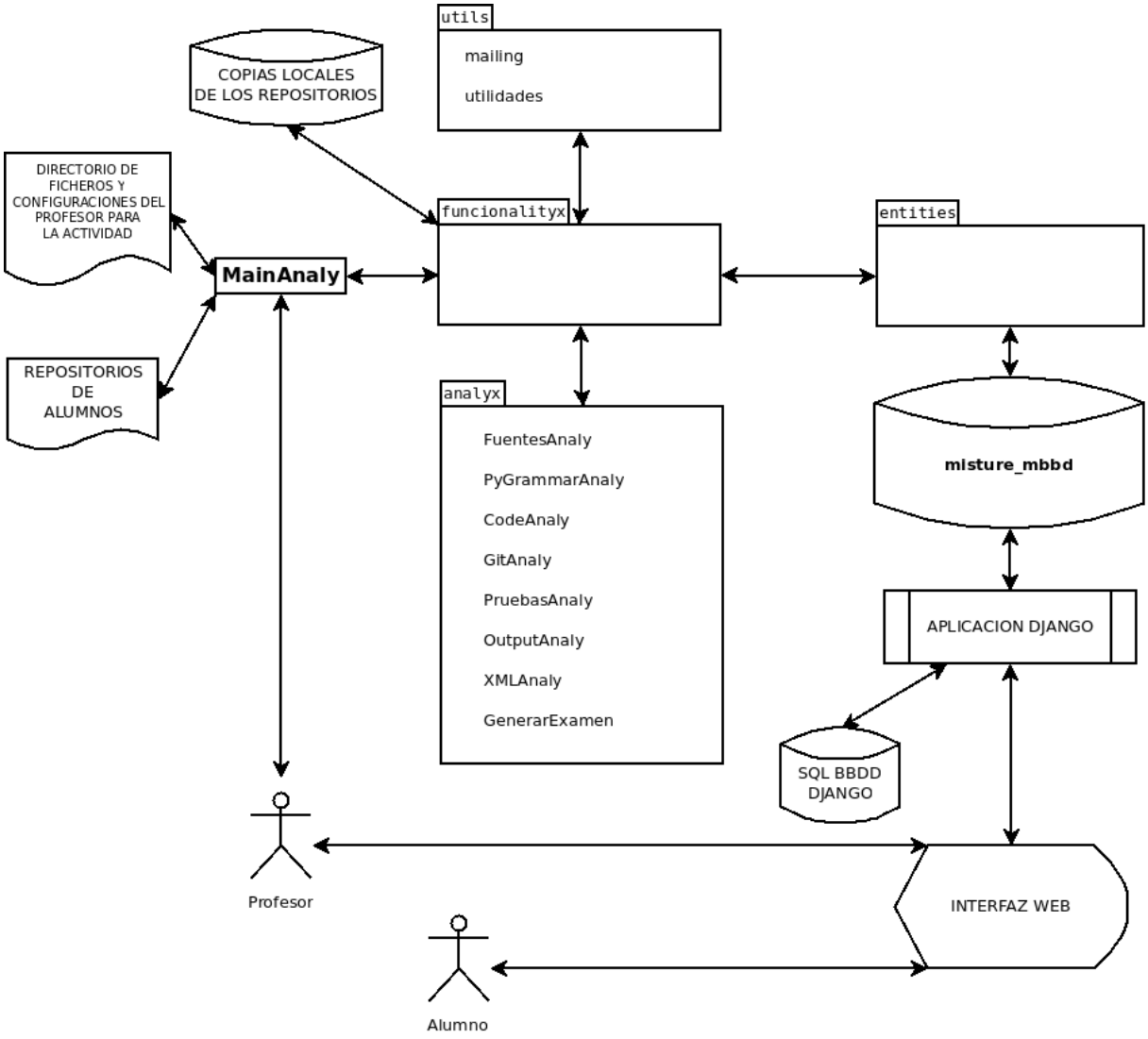


Figura 4.5: Relación entre los paquetes y elementos en esta versión de Misture

4.3.1. Módulo principal

Durante el transcurso de las iteraciones, se sigue manteniendo en el diseño el módulo principal como elemento de interacción con el profesor para la configuración, control y uso de la funcionalidad de Misture. La idea es que este módulo abstraiga el resto del sistema Misture y únicamente pueda usarse con la interfaz que proporcionemos a través del paquete *funcionalityx*, siendo sólo necesario editar la configuración y llamadas a los análisis deseados para realizar cualquier corrección.

Se mantiene la necesidad de disponer de los listados de alumnos y de la ubicación o url de los repositorios, así como de las tuplas con las pruebas de caja negra a ejecutar. Además, se introduce como novedad la definición en la configuración de un directorio del profesor especifico para cada corrección, donde se depositan los ficheros externos necesarios para las funcionalidades como las preguntas de examen, ficheros de entrada u otros módulos, por ejemplo la implementación de un cliente o servidor para usarlo en las pruebas contra la implementación de los alumnos.

4.3.2. Paquete de entidades

En la primera versión de Misture, se implementó una pequeña jerarquía de clases, con relaciones entre sí, que modelaban las diferentes realidades dentro de la corrección, con sus datos y relaciones entre ellos: una actividad, una corrección de actividad sobre un repositorio, un error de estilo o código, etc.

El siguiente paso en el diseño, ha sido trasladar este modelado a BBDD, de forma que podamos almacenar y recuperar sistemáticamente la información, y disponer de toda la información para posteriores reportes o análisis.

Al establecer el diseño de este paso, se optó por una base de datos no relacional como MongoDB, ya que para nuestro desarrollo incremental era más ágil y menos problemático no tener la exigencia de cumplir con un esquema fijo en los datos en todo momento.

Por otra parte, para poder centrarnos en la lógica de negocio de Misture, y abstraernos lo máximo posible de los accesos a datos, optamos también por el uso de un ODM, eligiendo *PyMODM*. De esta forma, podemos mantener la lógica de nuestro universo de objetos Misture y ampliarla con mayor facilidad.

Asimismo, *PyMODM* nos permite establecer el tipo de sus campos y otras restricciones, además de implementar validaciones como vemos en las figuras. Estas siempre se realizan a nivel de aplicación y no en BBDD.

```
class GitUser(MongoModel):
    name = CharField()
    email = CharField()

    def clean(self):
        import validators
        try:
            return validators.email(self.email)
        except validators.ValidationFailure:
            raise ValidationError(__class__.__name__ + '.email no valido')
```

```
    class Meta:
        indexes = [
            # Indice que equivale a un unique compuesto por name-email
            IndexModel([('actividad', ASCENDING), ('email', ASCENDING)],
                       unique=True, name='gituser_name_email_pk')
        ]
```

Figura 4.6: Un objeto Misture como documentos de una colección en MongoDB vía ODM

```
class Udfuentecoor(EmbeddedMongoModel):
    filaini = IntegerField()
    filafin = IntegerField()
    col = IntegerField()
    pos = IntegerField()
```

Figura 4.7: Representación de un documento sin colección, embebido en otro documento

4.3.3. Paquete de funcionalidades

Como ya se ha introducido, este paquete *funtionalityx* se encarga de implementar las funcionalidades de Misture a alto nivel, delegando las diferentes tareas que conforman un análisis a los módulos especializados del paquete *analyx*, estructurando los datos recibidos en los objetos del ODM para su guardado en las colecciones de MongoDB, y atendiendo a las llamadas que recibe de la interfaz que sirve al módulo principal.

Las funcionalidades que podemos invocar, son las que se detallan a continuación.

- Análisis de ficheros del repositorio.
- Análisis de elementos del código Python.

- Análisis de estilo y calidad del código.
- Análisis de repositorio Git.
- Análisis de XML y trazas de Wireshark.
- Generación de examen.

4.3.4. Paquete de análisis

En el diseño de las siguientes iteraciones de desarrollo, se mejoraron y ampliaron las funcionalidades de los análisis, buscando obtener funcionalidades a partir de librerías Python para Git, Pep8, u otras alternativas, lo que nos facilitaba el acceso a más opciones y datos.

Esto, junto con el resto de cambios comentados, justificó la reestructuración del proyecto, volcando dentro de un paquete los módulos de los diferentes análisis, aunando las viejas y nuevas funcionalidades implementadas.

A continuación los enumeramos.

Módulo de análisis de ficheros entregados

Con el objetivo de ampliar el análisis del contenido de la entrega, se crea un nuevo módulo, *FuentesAnaly*, que analiza el árbol del proyecto entregado, sin limitarse a evaluar un listado cerrado de ficheros, funcionalidad que mantiene el modulo *PyFichAnaly*.

En este módulo, se lee todo el árbol del directorio, almacenándose para su posterior guardado estructurado en la BBDD.

Asimismo, nos servimos de la herramienta *Sloccount*, ejecutado sobre el directorio del repositorio del alumno, para obtener parámetros de lenguaje, cantidad de líneas de código, de los ficheros fuente que se detecten. Con esos datos junto a la extensión del fichero, le asociamos el tipo de fichero.

```
$ sloccount --details --follow --duplicates <path_directorio>
```

Todos estos datos, se guardaran debidamente relacionados en la colección **Udfile** de nuestra BBDD.

Módulo de análisis de GIT

Para el desarrollo de una funcionalidad de análisis del repositorio Git más rica, se implementó un nuevo módulo *GitAnaly*, que permite a través de la introducción de la librería de Python Pygit2 el manejo de repositorios y la lectura de la estructura y contenidos de éste.

Este módulo proporcionamos, por una parte, funcionalidad para descargar, copiar o clonar los repositorios y puedan realizarse el resto de operaciones y análisis. Por otra parte, aprovechando la API de Pygit2, implementamos funcionalidad para extraer la información de las ramas locales y remotas. También se implementa otra funcionalidad la cual nos extrae en orden topológico la sucesión entre dos commits cuyo sha-1 se índique, o entre HEAD y el primero en su defecto. Se analizan todos los datos de un *commit*, disponiendo en este caso de una información más completa.

Entre esta información adicional, se extraen ciertas estadísticas para su almacenamiento:

- Frecuencia entre los *commits*.
- Extracción de palabras clave de un *commit*: se extraen la lista de palabras clave resultante de normalizar el comentario, suprimir las palabras consideradas de un listado de irrelevantes.

```
M_GIT_COMMENT_EXC_SPA = (
    'a', 'y', 'de', 'que', 'ni', 'la', 'sin', 'con',
    'from', 'pull', 'merge', 'request', 'para',
    'programa', 'patch', 'actualizar', '', 'del', 'un', 'en', 'el')
M_GIT_COMMENT_VIP_SPA = ('rtp',)
M_GIT_COMMENT_NORM = {
    'á': 'a', 'é': 'e', 'í': 'i', 'ó': 'o', 'ú': 'u',
    'Á': 'A', 'É': 'E', 'Í': 'I', 'Ó': 'O', 'Ú': 'U'
}
```

Figura 4.8: Parámetros para la extracción de significantes de comentarios de *commits*

Toda la información extraída según cada funcionalidad, se proporciona estructurada y los *commits* relacionados entre sí para que pueda ser todo almacenado en BBDD.

Módulo de análisis de código Python

En el análisis de los elementos del código fuente, introducimos en el diseño las funcionalidades que aportan la librería estándar *Ast* y la librería externa *Astor*, cuya función es la de procesar árboles de gramática abstracta, en Python.

Con ellas, podemos recuperar todos los elementos de código y sus propiedades en una estructura de árbol. Para la funcionalidad de este módulo, ambas librerias nos simplifican la implementación y abre nuevas posibilidades a futuras automatizaciones de Misture. *Astor* nos proporciona algunas capacidades adicionales para manipular estos árboles y convertir el árbol a otras estructuras, incluyendo su impresión a texto de nodos de código.

Esta funcionalidad la añadimos al módulo *PyGrammarAnaly*, para extraer datos por módulo sobre sus clases, métodos, línea y nombre, y pueda volcarse en la colección **UdFuente**. No obstante, mantenemos la anterior funcionalidad de la clase *PyModuleCont* y la trasladamos a este módulo, para su uso en prácticas pequeñas o sobre versiones antiguas de Python.

Módulo de análisis de estilo y calidad de código

En la implementación se mantuvo en gran medida la funcionalidad del módulo *CodeAnaly*. Se sustituyeron las llamadas a las utilidades por sus librerías equivalentes en Python. Se consiguió una pequeña mejora en la extracción de la información de Pylint obteniendo la salida de los errores en JSON.

La principal novedad, es la introducción de una nueva utilidad de análisis de *Flake8*, que combina, el análisis de estilo de *Pep8* con el análisis estático de *Pyflakes*, similar a *Pylint*. Asimismo, permite la instalación de otros plugins para ampliar el análisis. Por lo que extendemos, con los paquetes *McCabe* y *Hackings*, aumentando la cantidad de comprobaciones y códigos de error disponibles.

Módulos de análisis de XML y trazas Wireshark

En el análisis de los *scripts* semiautomáticos de las experiencias piloto, comentamos que existía un análisis de ficheros XML, que consistía en implementar un pequeño parser SAX específico para la corrección que comprueba la validación deseada sobre los elementos XML.

En esta versión, se ha empezado a introducir una aproximación mediante análisis del árbol DOM, con ayuda de la librería Beautiful Soup 4, que genera dicho árbol y nos proporciona un potente conjunto de funciones para navegar.

Gracias a esto, hemos implementado un pequeño analizador dedicado a PDML que obtiene algunas estadísticas de las trazas. Como por ejemplo, para extraer la composición de tramas de los paquetes de una trama.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="pdml2html.xsl"?>
<!-- You can find pdml2html.xsl in /usr/share/wireshark or at http://anonsvn.wireshark.org/trunk/wireshark/pdml2html.xsl. -->
<pdml version="0" creator="wireshark/1.10.6" time="Wed May 10 18:10:59 2017" capture_file="llamada.libpcap">
<packet>
  <proto name="geninfo" pos="0" showname="General information" size="100">
    <field name="num" pos="0" show="1" showname="Number" value="1" size="100"/>
    <field name="len" pos="0" show="100" showname="Frame Length" value="64" size="100"/>
    <field name="caplen" pos="0" show="100" showname="Captured Length" value="64" size="100"/>
    <field name="timestamp" pos="0" show="Jan  9, 2015 13:38:12.432193000 CET" showname="Captured Time" value="1420807092.432
  </proto>
  <proto name="frame" showname="Frame 1: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 0" size="10
    <field name="frame.interface_id" showname="Interface id: 0" size="0" pos="0" show="0"/>
    <field name="frame.encap_type" showname="Encapsulation type: Ethernet (1)" size="0" pos="0" show="1"/>
    <field name="frame.time" showname="Arrival Time: Jan  9, 2015 13:38:12.432193000 CET" size="0" pos="0" show="Jan  9, 2015
    <field name="frame.offset_shift" showname="Time shift for this packet: 0.000000000 seconds" size="0" pos="0" show="0.0000
    <field name="frame.time_epoch" showname="Epoch Time: 1420807092.432193000 seconds" size="0" pos="0" show="1420807092.4321
    <field name="frame.time_delta" showname="Time delta from previous captured frame: 0.000000000 seconds" size="0" pos="0" s
    <field name="frame.time_delta_displayed" showname="Time delta from previous displayed frame: 0.000000000 seconds" size="0
    <field name="frame.time_relative" showname="Time since reference or first frame: 0.000000000 seconds" size="0" pos="0" sh
    <field name="frame.number" showname="Frame Number: 1" size="0" pos="0" show="1"/>
    <field name="frame.len" showname="Frame Length: 100 bytes (800 bits)" size="0" pos="0" show="100"/>
    <field name="frame.cap_len" showname="Capture Length: 100 bytes (800 bits)" size="0" pos="0" show="100"/>
    <field name="frame.marked" showname="Frame is marked: False" size="0" pos="0" show="0"/>
    <field name="frame.ignored" showname="Frame is ignored: False" size="0" pos="0" show="0"/>
    <field name="frame.protocols" showname="Protocols in frame: eth:ip:udp:sip" size="0" pos="0" show="eth:ip:udp:sip"/>
  </proto>
  <proto name="eth" showname="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
    <field name="eth.dst" showname="Destination: 00:00:00_00:00:00 (00:00:00:00:00:00)" size="6" pos="0" show="00:00:00:00:00
      <field name="eth.addr" showname="Address: 00:00:00_00:00:00 (00:00:00:00:00:00)" size="6" pos="0" show="00:00:00:00:00:
      <field name="eth.lg" showname=".... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)" size=
      <field name="eth.ig" showname=".... ...0 .... .... .... .... = IG bit: Individual address (unicast)" size="3" pos="0" s
    </field>
    <field name="eth.src" showname="Source: 00:00:00_00:00:00 (00:00:00:00:00:00)" size="6" pos="6" show="00:00:00:00:00:00"
      <field name="eth.addr" showname="Address: 00:00:00_00:00:00 (00:00:00:00:00:00)" size="6" pos="6" show="00:00:00:00:00:
      <field name="eth.lg" showname=".... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)" size=
      <field name="eth.ig" showname=".... ...0 .... .... .... .... = IG bit: Individual address (unicast)" size="3" pos="6" s
    </field>
    <field name="eth.type" showname="Type: IP (0x0800)" size="2" pos="12" show="0x0800" value="0800"/>
  </proto>
```

```
LA TRAZA llamada.pdml TIEÑE LA SIGUIENTE COMPOSICION
Paquetes 35
    malformed   10
    fake-field-wrapper  15
    sdp 2
    eth 35
    udp 35
    sip 16
    ip  35
```

Figura 4.9: El PDML de la traza son elementos *packet* con elementos *proto* en su interior.

4.3.5. Generación de examen

En el diseño, se incluye este módulo que permite la generación y carga para la corrección de cada alumno, de un conjunto de preguntas acerca del código y de los conceptos que el profesor quiera poner a prueba. Para ello, consideramos dos fuentes:

- Se carga un fichero JSON con las preguntas desde la ruta que se le indica como parámetro. La estructura de este JSON es una lista de objetos pregunta con los elementos enunciado, pregunta, tipo, opciones, resultados, etc.
- A partir del análisis del código, se obtienen algunos snippets del código de las fuentes Python, pertenecientes a correcciones de los diferentes alumnos de una misma actividad.

A partir de esas dos fuentes, y en función del tipo de pregunta, se rellenan documentos en la colección **CuestionExamen** de la BBDD para cada una de las correcciones. En el caso de las preguntas generadas a partir de snippets, se generan preguntas con opciones si o no sobre la autoría o sobre al nombre del fichero fuente al que pertenece.

Posteriormente, en caso de cargarse las respuestas en los documentos de **CuestionExamen**, podemos validarlas contra la respuesta esperada.

4.4. Aplicación examen Django

Una aplicación de examen recoge información a través de la base de datos MongoDB de Misture con la carga de las preguntas generadas.

Para los alumnos cuyos logines se encuentran en el conjunto de correcciones de la actividad, se activa la opción de contestar a las preguntas en forma de formularios Esta queda marcada como respondida una vez se pulse al botón de confirmación y queda asimismo registrado el momento en que fue contestada.

4.5. 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, se buscó os una librería ODM -Object Document Mapper- para MongoDB en Python, que permitiera 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 se utilizó *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.

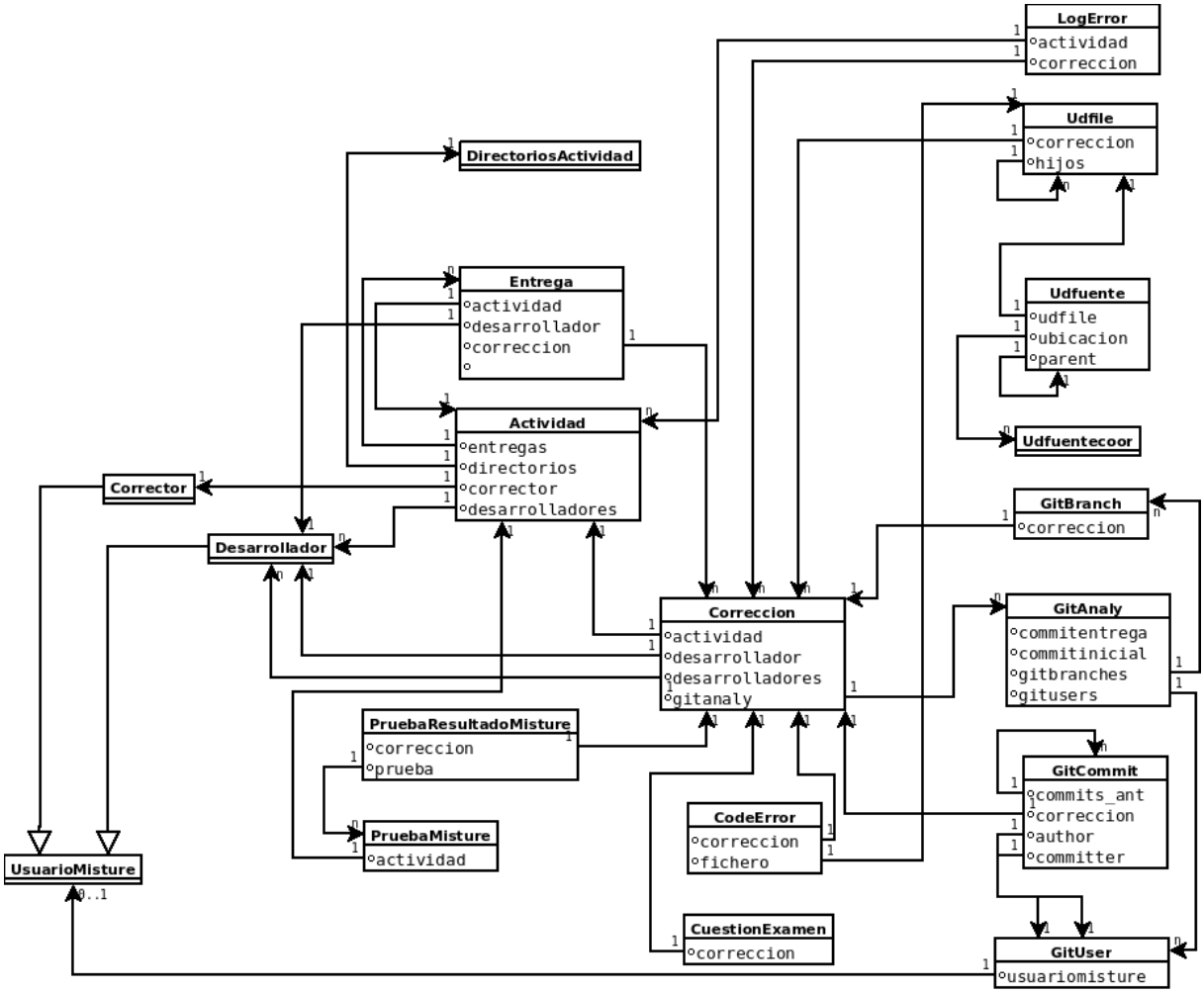


Figura 4.10: Representación de BBDD incluida en Misture

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.

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
- snippet

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

- filaini
- filafin
- col
- pos

CodeError: modela un error de estilo

- correccion: referencia a **Correccion**
- fichero
- fila
- columna
- tipo
- codigo
- descripcion

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**

PruebaMisture representa la definición de una prueba de caja negra.

- actividad: referencia a **Actividad**.
- titulo
- descripcion
- comando
- salida

PruebaResultadoMisture representa el resultado de una ejecución de prueba sobre una corrección de una entrega particular.

- correccion: referencia a **Corrección**.
- prueba: referencia a **PruebaMisture**
- tipo_salida
- path_salida
- espruebavalida

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 recoja 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 en la automatización a errores por parte de los alumnos en los códigos o en los ficheros entregables.
- 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.
- Mejorar 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.

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, se produce en el hecho de disponer de los datos de los análisis almacenados sistemáticamente con una mayor riqueza y relacionados en un medio persistente -BBDD-.

En cambio, el permitir tolerancia en los procesos automáticos frente a pequeños errores de los alumnos ha sido uno de los hitos más complicados de cumplir.

En esa dirección 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. Sin embargo, 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 tienen los alumnos en sus entregas.

Por ejemplo, en casos en los que no se define estrictamente en el enunciado de la actividad propuesta, o el alumno comete errores, sobre la forma de usar o leer los parámetros de los *scripts*. En este caso concreto esto puede 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. También 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 para abarcar escenarios de corrección menos específicos. Tampoco lo es alcanzar la automatización plena y desatendida de la herramienta, al menos en los procesos que hay entre la configuración de Misture y la lectura de informes y reporte.

Por ejemplo, una ejecución de código de caja negra es difícil de adecuar para probar una práctica desarrollada en un *framework* de aplicaciones web como Django, empleada por los alumnos en asignaturas de cursos más avanzados.

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 de corrección lo más generales posibles debe estar muy bien estudiada en sus elementos o circunstancias para evitar encontrarse ante un caso complicado de manejar y de controlar.

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

También puedo considerar que he aprendido a evaluar mejor que proyectos o APIs debo elegir, y he 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, 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 mi, 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 -SAT-.

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 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 no era desconocido 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 apartado, se describen las posibles mejoras o líneas futuras que permitan un mayor desarrollo del proyecto obteniendo mejoras en el mismo.

Dado que personalmente observé 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 para poderse configurar desde esta interfaz 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.
- Integrar capacidades propias para detección de plagio.
- Otra posibilidad a explorar es integrar la funcionalidad de MOSS en el propio Misture, debido a la existencia de *scripts* cliente y de librerías para la visualización y análisis de los resultados del análisis.

Apéndice A

Instalación y Uso

Con el sistema Misture propuesto, si se desea efectuar una corrección, serían necesarios:

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 módulos de corrección.

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

- Asimismo, se pueden ajustar otras constantes como el listado de errores o tipo de errores de Pep8, Pylint o Hackings que se desean ignorar en un análisis de estilo y código.
- También, en caso de activar la comprobación de XML o PDML, se indicarían que comprobaciones, entre las implementadas en Misture, 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

A continuación se especifican las dependencias necesarias para la ejecución del código de Misture.

En cuanto a librerías Python, si no se indica lo contrario, se aconseja realizar con gestor de paquetes de Python llamado 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, sin olvidar que estas precisan de otras dependencias requeridas que serán instaladas conjuntamente.

Además, 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.

```
$ sudo pip install virtualenv
$ cd <directorio_ubicar_env>
$ virtualenv ENV
$ source ENV/bin/activate
```

Veremos que aparece (ENV) al principio de nuestro prompt. En ese entorno se pueden instalar en Python todas las librerías necesarias, y lanzar el proyecto. Para abandonar este entorno y volver al habitual, invocamos, la siguiente instrucción, desapareciendo la cadena (ENV) de nuestro prompt.

```
$ deactivate
```

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 entorno Python 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 Pymongo y el ODM PyMODM de Python: PyMongo 3.4, siendo válida cualquier versión con numeración mayor o igual a la de MongoDB empleada. Usar la versión 0.4.0 o superior de PyMODM, teniendo en cuenta en la documentación de esta librería, que se indica que esta probado 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 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 esta probado con la versión 0.25.0
- Librería Beautiful Soup. Se debe instalar Beautiful Soup 4. Automáticamente se instalan el resto de dependencias.

- Librería Python de la herramienta Flake8 Se ha empleado la versión reciente 3.3.0. Automáticamente se instalan el resto de dependencias.
- Librería Hacking (Extensión para Flake8): Empleada la versión 0.13
- Librería Pep8: Empleada la versión 1.7
- Librería Pylint: Empleada la versión 1.7.1
- Librería Validators: Se ha empleado la versión 0.11.3
- Librería Astor: Versión >=0.5

Bibliografía

[1] Gregorio Robles y Jesús M. González-Barahona (2013). Mining student repositories to gain learning analytics

[2] Robles, G., González-Barahona J. M., Izquierdo-Cortazar, D. y Herraiz, I. (2009). Tools for the Study of the Usual Data Sources found in Libre Software Projects.

[3] Web de las conferencias de MSR:
`http://2017.msrconf.org/`

[4] Pro Git book. Scott Chacon and Ben Straub.
`https://git-scm.com/book/es/v1`

[5] Guía de estilo de Python PEP8.
`https://www.python.org/dev/peps/pep-0008/`

[6] Análisis estático de código.
`http://raulexposito.com/documentos/analisis-estatico-codigo/`

[7] Referencia y librerías estándar de Python 2.
`https://docs.python.org/2/`

[8] Referencia y librerías estándar de Python 3
`https://docs.python.org/3/`

[9] Documentación de MongoDB
`https://docs.mongodb.com/manual/`

[10] Diferencias entre SQL y MongoDB

https://docs.mongodb.com/manual/reference/sql-comparison/

[11] Documentación de la librería Pymongo

https://api.mongodb.com/python/current/

[12] Documentación de la librería PyMODM

https://github.com/mongodb/pymodm

[13] Documentación de Pygit2

http://www.pygit2.org/

[14] Documentación de Flake8

https://flake8.readthedocs.io/en/latest/index.html

https://gitlab.com/pycqa/flake8

[15] Documentación de la utilidad Pep8 -Pycodestyle-.

http://pep8.readthedocs.io/en/release-1.7.x/

https://github.com/PyCQA/pycodestyle

[16] Documentación de Pylint

https://pylint.readthedocs.io/en/latest/

[17] Documentación de Astor.

https://github.com/berkerpeksag/astor

[18] Web del IDE PyCharm

https://www.jetbrains.com/pycharm/

[19] Web de la aplicación Robomongo.

https://robomongo.org/