



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**Evaluación de simulaciones de
proyectos en GitHub
orientadas a docencia.**



Presentado por Licinio Fernández Maurelo
en Universidad de Burgos — 6 de junio
de 2023

Tutor: Dr. Carlos López Nozal



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



D. Dr. Carlos López Nozal, profesor del Departamento de Ingeniería Informática, Área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Licinio Fernández Maurelo, con DNI 04223955M, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado Evaluación de simulaciones de proyectos en GitHub orientadas a docencia.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 6 de junio de 2023

Vº. Bº. del Tutor:

D. Dr. Carlos López Nozal

Resumen

Los laboratorios virtuales se han convertido en una alternativa cada vez más popular a las sesiones de laboratorio tradicionales en varios campos de estudio. Este proyecto se enfoca en laboratorios virtuales para la gestión ágil de proyectos y el control de versiones usando git. A pesar de la creciente popularidad de los laboratorios virtuales, su eficacia para promover los resultados del aprendizaje sigue siendo un tema de debate. Para mejorar el proceso de aprendizaje este proyecto implementa una herramienta de evaluación automática para ambos laboratorios virtuales.

Descriptores

laboratorio virtual, gestión de proyectos ágil, control de versiones, git, evaluación automática

Abstract

Virtual labs have become an increasingly popular alternative to traditional laboratory sessions in various fields of study. This project focus on virtual labs for agile project management and version control using git. Despite the increasing popularity of virtual labs, their effectiveness in promoting learning outcomes is still a matter of debate. In order to improve the whole learning process this project implements a tool for automatic assessment for both virtual labs.

Keywords

virtual lab, agile project management, version control, git, automatic assessment

Índice general

Índice general	iii
Índice de figuras	v
Índice de tablas	vii
Introducción	1
Objetivos del proyecto	5
Conceptos teóricos	7
3.1. Evaluación de laboratorios virtuales	7
3.2. Conceptos de la gestión de proyectos ágiles utilizados en el laboratorio virtual	8
3.3. Conceptos de control de versiones utilizados en el laboratorio virtual	9
3.4. Evaluación del laboratorio virtual de gestión de proyectos ágiles.	11
3.5. Evaluación del laboratorio virtual de control de versiones . .	12
3.6. Similitud	12
Técnicas y herramientas	17
4.1. Metodología de desarrollo del proyecto	17
4.2. Unit testing	18
4.3. Integration testing	19
4.4. JUnit	19
4.5. Test-driven development (TDD)	19

4.6. User acceptance testing (UAT), Acceptance Test-Driven Development (ATDD)	21
4.7. Behaviour driven development (BDD)	21
4.8. Cucumber	22
4.9. Arquitectura hexagonal	25
4.10. Integración continua. GitHub Actions	26
4.11. Insomnia	28
4.12. Open Feign	29
4.13. Maven	29
4.14. Spring Boot	30
4.15. Thymeleaf	31
4.16. Bootstrap	31
4.17. Sonarcloud	32
4.18. Chat GPT	32
4.19. Lombok	35
Aspectos relevantes del desarrollo del proyecto	37
5.1. Integración con GitHub	37
5.2. Integración con Zenhub	39
5.3. Comunicación entre alumno y tutor mediante el flujo de trabajo con GitHub	40
5.4. Trazabilidad en GitHub de la aplicación de BDD y TDD	42
5.5. Interpretación de resultados obtenidos en la evaluación del laboratorio virtual de control de versiones	44
5.6. Discrepancias entre la rúbrica de la evaluación automática y la utilizada en la evaluación manual	45
5.7. Distribución y revisión de la aplicación	45
Trabajos relacionados	47
Conclusiones y Líneas de trabajo futuras	49
7.1. Conclusiones	49
7.2. Líneas de trabajo futuras	50
Bibliografía	51

Índice de figuras

1.1. Descripción cuantitativa y cualitativa de la simulación de un sprint/iteración	2
1.2. Gestión de tareas de un proyecto software: extracto de rúbrica para la evaluación	3
4.1. Detalle de sprints en estado cerrado	17
4.2. Detalle de la tarea de inclusión de Cucumber en el proyecto	18
4.3. Ciclo de desarrollo TDD	20
4.4. Ciclo de vida BDD	22
4.5. Relación de los ciclos de vida de desarrollo BDD y TDD	22
4.6. Ficheros de features de este trabajo	24
4.7. Capas de la arquitectura hexagonal	25
4.8. Definición de acciones de integración continua en GitHub actions	27
4.9. Histórico de ejecuciones de flujos de acción	28
4.10. Captura de la herramienta Insomnia	28
4.11. Implementación de la operación de obtención de un commit utilizando Open Feign	29
4.12. Uso de componente bootstrap navbar y visualización	32
4.13. Petición de implementación de algoritmo a Chat GPT	33
4.14. Petición de implementación de tests unitarios para la implementación Java del algoritmo de Jaro-Winkler	34
4.15. Análisis Sonar Cloud de la implementación Java del algoritmo de Jaro-Winkler	34
5.1. Comparativa de la información de gestión de tareas obtenida de forma manual y automática	39
5.2. Captura cierre de tarea referenciada en el commit	40
5.3. Creación de rama a partir de una tarea	40

5.4. Instrucciones para descargar la rama asociada a la tarea en el repositorio local	41
5.5. Notificación de cambios recientes en una rama	41
5.6. Detalle de pull request mergeada y cerrada	42
5.7. Detalle de la pull request en GitHub	43
5.8. Detalle del commit con mensaje WIP BDD red	43
5.9. Datos de entrada ejemplo para la evaluación del laboratorio virtual de control de versiones	44
5.10. Desalineamiento de commits	44
5.11. Ejecución y acceso desde un entorno local a la aplicación	46

Índice de tablas

Índice de fragmentos de código

4.1.	Ejemplo de sintaxis Gherkin	23
4.2.	Extracto de definición de pasos de una feature	24
4.3.	Clase Java convencional	35
4.4.	Clase Java utilizando Lombok	36

Introducción

En la actualidad, la enseñanza a distancia goza de gran importancia debido a ventajas inherentes como la deslocalización de profesores y estudiantes y la libertad de agenda de los estudiantes a la hora de seguir la formación.

Los centros de formación pueden hacer un uso más eficiente de los recursos educativos, lo que permite ampliar su capacidad de estudiantes al no estar sujetos a limitaciones propias de la enseñanza presencial como aulas y laboratorios.

Sin embargo, la enseñanza a distancia de materias relacionadas con la ciencia, la tecnología y la ingeniería presenta dificultades derivadas de la propia naturaleza de estas disciplinas: a menudo requieren prácticas en el laboratorio para dotar al estudiante de las competencias necesarias.

Históricamente, algunas de las necesidades especiales de los estudios universitarios de ingeniería no han sido bien cubiertas por los métodos de enseñanza online [1].

La emergencia de las tecnologías de la información ha permitido el desarrollo de laboratorios virtuales. Su acceso online y el software que los soporta permite a los estudiantes una reproducción cada vez más fiel del trabajo realizado en el laboratorio.

Los laboratorios virtuales son esenciales para que los estudiantes adquieran conocimientos prácticos que se convertirán en activos fundamentales en su carrera profesional [2].

En el desarrollo de laboratorios virtuales, uno de los focos de acción es facilitar la autoevaluación para mejorar la experiencia de aprendizaje de los estudiantes y reducir la carga de trabajo de los estudiantes. Una autoevaluación clara y concisa lleva al alumno a una reflexión sobre sus

propios errores, convirtiendo una evaluación formativa en una evaluación formadora [3].

La enseñanza de un modelo ágil de gestión de proyectos mediante GitHub, ZenHub y el sistema de control de versiones Git proporciona el marco de este trabajo de fin de grado. En concreto nos centraremos en la asignatura de Gestión de proyectos en el bloque correspondiente a la Gestión ágil (ver [guía docente](#)¹):

- Simulación de gestión de tareas de un proyecto software. El estudiante realiza una simulación de planificación de tareas de un proyecto software.
- Simulación de control de versiones de un proyecto open source. El estudiante simula la gestión de versiones de código utilizando un proyecto disponible en un repositorio público.

Los casos de estudio de simulación de ambas actividades proporcionan el detalle paso por paso de la simulación a realizar, como el que se muestra en la Figura 1.1 y, además, el estudiante cuenta con el repositorio original a modo de referencia.

Sprint 16	17 closed = 7	
	documentation + 4	1 issue con comment
6 días	feature + 2 testing + 4	
	bug	#164 Las imágenes no se posicionan correctamente (enlace + 8 commits)
53 puntos		
de	Todas asignados a	0 issue con tasklist
historia	David	
		3 issues con comments
	9 closed = 2	

Figura 1.1: Descripción cuantitativa y cualitativa de la simulación de un sprint/iteración

El producto entregable es la clonación del repositorio o url del repositorio con el caso de estudio de simulación. Para la evaluación de cada uno de los casos de estudio de simulación, existe una rúbrica de evaluación que se muestra en la Figura 1.2 que actualmente es aplicada de forma manual en base al producto entregado.

¹Guía docente de la asignatura Gestión de proyectos curso 2022-2023 : https://ubuvirtual.ubu.es/mod/guiadocente/get_guiadocente.php?asignatura=6366&cursoacademico=2022

Valoración del trabajo en equipo en Github	Participación menos del 50% de los miembros del equipo en las tareas 0 points	Participación del 100% de los miembros del equipo en las tareas 2 points	
Estimaciones temporales	Existen más del 75% de elementos de Scrum sin asignar tiempos 0 points	Se aplican tiempos a todos los elementos Scrum de la aplicación pero se desvian mucho de la realidad 1 points	Se aplican tiempos a todos los elementos Scrum planificados. Los tiempos se aproximan a la realidad. 2 points

Figura 1.2: Gestión de tareas de un proyecto software: extracto de rúbrica para la evaluación

La evaluación de ambos casos de estudio de simulación puede automatizarse en gran medida. La información de las planificación de un proyecto en GitHub y en ZenHub, así como la información del control de versiones de GitHub, es accesible vía API.

A partir del contraste de la información del proyecto referencia y el proyecto de la simulación podemos dar contestación a gran parte de los elementos de la rúbrica. Del mismo modo, podemos explotar esta comparación para proporcionar retroalimentación al estudiante de aquellos elementos de la simulación que concuerdan o discrepan con el original.

La evaluación automática de las competencias en la gestión de tareas de tareas de un proyecto y del control de versiones es también de utilidad fuera del ámbito académico. Por ejemplo, podría utilizarse en el proceso de evaluación de competencias técnicas en el mundo empresarial, en las áreas siguientes:

- Procesos de selección.
- Evaluaciones internas de personal de la compañía.
- Formación.

En este trabajo, dado que surge de un dominio académico, se abordará la evaluación de los casos de estudio desde una perspectiva académica. Alternativamente, podría también haberse abordado desde la perspectiva de un proceso de control de calidad.

Objetivos del proyecto

La realización de este proyecto tiene como objetivo principal el desarrollo de un software que permita automatizar la evaluación y retroalimentación de los siguientes casos de estudio de simulación planteados como laboratorio virtual en la asignatura gestión de proyectos del grado en ingeniería informática:

1. Simulación de gestión de tareas de un proyecto software.
2. Simulación de control de versiones de un proyecto open source.

Para acometer lo anterior el software desarrollado deberá implementar las siguientes funcionalidades:

1. Obtención de información en un intervalo temporal de la gestión de tareas de un proyecto alojado en GitHub con la extensión ZenHub activada:
 - a) Sprints.
 - 1) Título.
 - 2) Fecha de finalización.
 - 3) Estado.
 - b) Tareas.
 - 1) Descripción.
 - 2) Etiquetas.
 - 3) Puntos de historia.

- 4) Comentarios.
 - 5) Estado.
2. Obtención de información en un intervalo temporal del control de versiones de un proyecto alojado en GitHub:
 - a) Commits.
 - 1) Identificador único (SHA, Simple Hashing Algorithm).
 - 2) Fecha.
 - 3) Autor.
 3. Comparación de la información de la gestión de tareas de dos proyectos.
 4. Comparación de la información del control de versiones de dos proyectos.
 5. Aplicación de rúbrica de evaluación sobre:
 - a) La información de gestión de tareas y del control de versiones de un repositorio.
 - b) La comparación de la información de gestión de tareas y de control de versiones de dos repositorios, el original y el resultante de la simulación.

Los objetivos técnicos del desarrollo son:

1. Uso de técnicas de gestión de proyectos ágiles.
2. Cumplimiento de estándares de calidad de software y aseguramiento de calidad mediante análisis automatizados con Sonarqube.
3. Integración con GitHub.
4. Integración con ZenHub.
5. Uso de una arquitectura hexagonal [4] que desacople la funcionalidad desarrollada de los orígenes de datos. Esto facilitará la extensibilidad de la funcionalidad simplificando la integración con otros gestores de tareas y gestores de repositorios Git.

Conceptos teóricos

3.1. Evaluación de laboratorios virtuales.

Laboratorios virtuales.

El objetivo principal de este trabajo es implementar la automatización de la evaluación de los laboratorios virtuales siguientes:

- laboratorio virtual de gestión de proyectos ágiles.
- laboratorio virtual de control de versiones.

Estos laboratorios virtuales presentan una serie de características comunes en sus planteamientos.

En ambos se presenta un **caso de estudio** que el profesor utiliza como referencia de buenas prácticas para el aprendizaje.

Los alumnos deben realizar una **simulación** del caso de estudio con condiciones específicas añadidas como el trabajo en equipo.

Evaluación.

Como instrumento de evaluación de cada uno de los laboratorios virtuales se utiliza una **rúbrica**.

Cada rúbrica utiliza una serie de **criterios de evaluación** para determinar el grado de aprendizaje de los alumnos.

3.2. Conceptos de la gestión de proyectos ágiles utilizados en el laboratorio virtual.

Se presentan a continuación de forma breve conceptos de la gestión de proyectos ágiles utilizados en el laboratorio virtual, consultar [5] para obtener un conocimiento más detallado de los conceptos.

Historia de usuario

Es una descripción informal de una funcionalidad del sistema a desarrollar. Se documentan desde la perspectiva del usuario final del sistema.

Puntos de historia de usuario

Es una métrica utilizada por los miembros del equipo de proyecto para expresar el esfuerzo estimado requerido para completar una tarea. En la estimación se considera:

1. Complejidad de la tarea.
2. Cantidad de trabajo de la tarea.
3. Riesgos, incertidumbres de la tarea a realizar.

Sprint/Milestone/Hito

Es un periodo de tiempo durante el cual se realiza un trabajo específico, definido en las distintas tareas que componen el sprint. En el laboratorio virtual es de relevancia la siguiente información:

1. Fecha de inicio.
2. Fecha de fin.
3. Puntos de historia de usuario: total de puntos de historia de usuario de las tareas incluidas en el sprint.

Tarea/Issue.

Cada historia de usuario es dividida en tareas: unidades de trabajo individuales. En el laboratorio virtual se plantea especificar la siguiente información:

1. Descripción de la tarea.
2. Comentarios de una tarea: Una tarea puede incluir comentarios, cada comentario tiene un autor.
3. Miembro del equipo al que se le asigna la tarea.
4. Puntos de historia de usuario.
5. Tipos de tarea:
 - a) Issue: tarea de propósito general.
 - b) Bug: incidencia.
6. Estados de una tarea:
 - a) Open/abierta.
 - b) Closed/cerrada.
7. Etiquetas: permiten la clasificación de una tarea en diversas categorías. Existen etiquetas predefinidas y etiquetas personalizadas definidas por el usuario. A efectos de la simulación se consideran las siguientes etiquetas predefinidas:
 - a) Feature/funcionalidad.
 - b) Testing/pruebas.
 - c) Documentation/documentación.

3.3. Conceptos de control de versiones utilizados en el laboratorio virtual

Respecto a el control de versiones, se explican a continuación los conceptos utilizados en este trabajo.

Control de versiones

El control de versiones consiste en la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo [6].

En el contexto de las tecnologías de información, este control de versiones se realiza mediante herramientas que permiten gestionar activos de elementos software como:

- Ficheros de código fuente a partir de los que se generan programas informáticos.
- Ficheros de configuración.
- Otros tipos de ficheros con formato binario empleados en el desarrollo de programas informáticos, como por ejemplo imágenes.

En la actualidad el uso de herramientas de control de versiones se extiende fuera de la industria de software a otras industrias, como por ejemplo a la industria de la edición y publicación de libros.

De entre todas la herramientas de control de versiones, Git, es el estándar de facto.

Git

En este trabajo se utilizan los siguientes términos pertenecientes al glosario de términos de Git [7] .

- Repositorio: colección de enlaces a objetos y base de datos de objetos en los que se almacenan los elementos a gestionar.
- Rama/Branch: línea de desarrollo. Cada rama mantiene su propia colección de enlaces.
- Commit: Un punto individual de la historia de Git. Entre otros datos, contiene información de:
 - Autor.
 - Fecha y hora del commit.
 - Comentarios asociados.
 - Ficheros afectados.

GitHub

Es un servicio de alojamiento en internet de repositorios Git.

3.4. Evaluación del laboratorio virtual de gestión de proyectos ágiles.

Valoración del trabajo en equipo en Github

Se basa en el porcentaje medio de participación del grupo de alumnos en las tareas. La realización de la práctica esperada es:

- Uno de los alumnos crea la tarea.
- Otro modifica la tarea. Se realizan modificaciones hasta llevar la tarea al estado final deseado con la sucesiva participación de todos los integrantes del grupo.

Estimaciones temporales

En el laboratorio virtual se propone crear un sprint ficticio a partir de las tareas abiertas del caso de estudio.

La evaluación de este criterio está basada en las características de las tareas que conforman el sprint ficticio:

- Establecer estimaciones temporales de cada tarea.
- Precisión de la estimación en relación a otros sprints del caso de estudio.

Aprendizaje de herramientas de gestión de tareas

Valora el conjunto de características informadas en cada una de las tareas de cara a determinar el grado de madurez de los alumnos en el uso de la herramienta de gestión de tareas.

Planificación de Sprint

Evalúa el sprint ficticio considerando:

- Las tareas del sprint están asignadas.
- Las tareas están correctamente descritas y son coherentes con la velocidad del proyecto.

3.5. Evaluación del laboratorio virtual de control de versiones

Valoración del trabajo en equipo en GitHub

Valora que los commits de la simulación se realicen alternativamente entre los miembros del equipo.

Simulación del control de versiones

Valora el porcentaje de commits de la simulación semejantes a los commits del caso de estudio.

Trazabilidad de las tareas con el sistema de control de versiones

Se valora el porcentaje de commits de la simulación que tienen establecida una tarea en la herramienta de gestión de proyectos ágiles.

En este caso, el laboratorio virtual propone que todos los commits deben tener esta información con independencia de cómo estén informados los commits en el caso de estudio.

3.6. Similitud

La evaluación de la simulación de laboratorios virtuales requiere determinar el grado de similitud de los resultados obtenidos.

Dada la naturaleza heterogénea de los elementos que conforman tareas y commits, se plantea el uso de funciones de similitud ponderadas para solventar esta problemática así como de un umbral de similitud.

Función de similitud

Una función de similitud es una herramienta matemática que nos ayuda a comparar dos objetos o conjuntos de datos para determinar cómo de similares o diferentes son [8].

La función de similitud toma dos entradas y produce un valor que refleja su similitud.

Se utilizan diferentes funciones de similitud según el tipo de datos que se comparan y su aplicación específica. En el contexto de este trabajo se utilizan las siguientes funciones de similitud:

1. Índice de Jaccard, para determinar la similitud de dos conjuntos.
2. Similitud de Jaro Winkler, para determinar la similitud de dos cadenas de caracteres.

Umbral de similitud

Una función de similitud devuelve valores continuos en el intervalo [0, 1]. Establecemos un **umbral de similitud** que será el valor mínimo de la función de similitud para el cual consideramos que los dos elementos son semejantes.

Índice de Jaccard

El índice de Jaccard se usa comúnmente en campos como el análisis de datos, la recuperación de información y el aprendizaje automático para comparar la similitud de dos conjuntos de datos [9]. Está definida por la expresión siguiente:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Donde $|A|$ representa la cardinalidad del conjunto A .

El valor resultante varía de 0 a 1, donde 0 indica que no hay similitud y 1 indica similitud total.

Distancia de Jaro Winkler

La distancia de Jaro-Winkler es una herramienta útil para comparar cadenas de texto y puede ayudar a identificar posibles coincidencias incluso cuando hay pequeños errores o diferencias en las cadenas [10].

La distancia se calcula en función del número de caracteres coincidentes entre las dos cadenas, así como del número de transposiciones de caracteres adyacentes. La distancia varía de 0 (coincidencia exacta) a 1 (sin caracteres coincidentes).

La distancia de Jaro-Winkler de las cadenas de caracteres s_1 y s_2 está determinada por la expresión:

$$d_{JW}(s_1, s_2) = \begin{cases} 0 & \text{si } m = 0 \\ 1 - \frac{1}{3} \left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m} \right) & \text{en otro caso} \end{cases}$$

donde m es el número de coincidencias de las dos cadenas de caracteres y t es el número de transposiciones de caracteres coincidentes adyacentes.

Similitud de Jaro-Winkler

La similitud de Jaro-Winkler varía de 0 (sin similitud) a 1 (similitud total.)

Para su cálculo se utiliza la distancia de Jaro-Winkler. Está determinada por la expresión:

$$s_{JW}(s_1, s_2) = 1 - d_{JW}(s_1, s_2)$$

Función de similitud combinada ponderada

En este trabajo, para determinar la similitud de tareas y commits se hará uso de una función de similitud combinación de la aplicación de varias funciones de similitud sobre características de interés de los objetos de estudio. Adicionalmente, cada función estará ponderada en el cálculo total.

La función de similitud $f(x)$ se define como:

$$f(x) = \sum_{i=1}^n w_i \cdot g_i(x)$$

donde w_i es el peso de la función $g_i(x)$, y n es el número de funciones que se combinan. La función ponderada es una forma de combinar múltiples funciones en una sola función, donde a cada función se le asigna un peso en función de su importancia o relevancia.

Función de similitud de tareas

Supongamos que tenemos dos tareas (issues) I_1 y I_2 que queremos comparar en función de su estado, etiquetas y características de la descripción (contiene una lista de tareas, contiene enlaces y contiene imágenes). Sean s_1 y s_2 el estado para I_1 y I_2 , respectivamente, y sean L_1 y L_2 los conjuntos de etiquetas en I_1 y I_2 , respectivamente. Sean D_1 y D_2 los conjuntos de características de la descripción en I_1 y I_2 , respectivamente. Podemos definir la función de similitud ponderada entre I_1 y I_2 como:

$$S(I_1, I_2) = w_1 \cdot sim(s_1, s_2) + w_2 \cdot sim(L_1, L_2) + w_3 \cdot sim(D_1, D_2)$$

donde $sim(s_1, s_2)$ es la similitud entre los estados de las tareas I_1 y I_2 , calculado usando el índice de Jaccard. $sim(L_1, L_2)$ es la similitud entre los conjuntos de etiquetas de las tareas I_1 y I_2 , calculada mediante el índice de Jaccard y $sim(D_1, D_2)$ es la similitud entre los conjuntos de características de la descripción de las tareas I_1 y I_2 , calculada mediante el índice de Jaccard.

Los pesos w_1 , w_2 y w_3 se asignan a estado, etiquetas y a características de la descripción de tareas, respectivamente, y se pueden ajustar según el peso de cada función.

Función de similitud de commits

Supongamos que tenemos dos commits C_1 y C_2 que queremos comparar en función de su mensaje y archivos modificados. Sean m_1 y m_2 los mensajes de commit para C_1 y C_2 , respectivamente, y sean F_1 y F_2 los conjuntos de archivos modificados en C_1 y C_2 , respectivamente. Podemos definir la función de similitud ponderada entre C_1 y C_2 como:

$$S(C_1, C_2) = w_1 \cdot sim(m_1, m_2) + w_2 \cdot sim(F_1, F_2)$$

donde $sim(m_1, m_2)$ es la similitud entre los mensajes de commit m_1 y m_2 , calculado usando la similitud de Jaro-Winkler, y $sim(F_1, F_2)$ es la

similitud entre los conjuntos de nombres de los archivos modificados F_1 y F_2 , calculada mediante el índice de Jaccard. Los pesos w_1 y w_2 se asignan al mensaje de commit y a los archivos de confirmación, respectivamente, y se pueden ajustar según el peso de cada función.

Técnicas y herramientas

4.1. Metodología de desarrollo del proyecto

Para el desarrollo de este trabajo se ha hecho uso de una metodología ágil gestión de proyectos.

El trabajo se ha dividido en once sprints de una duración media de dos semanas por sprint en los que se han realizado más de noventa tareas.

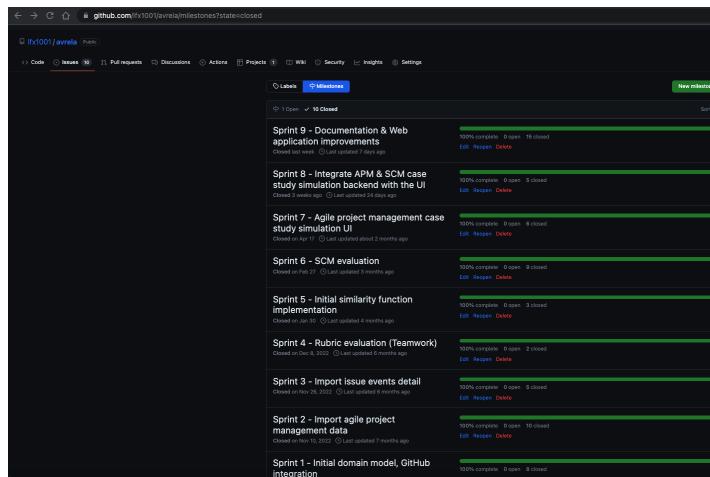


Figura 4.1: Detalle de sprints en estado cerrado

Todos los sprints y tareas están documentadas utilizando la herramienta de gestión de tareas de GitHub.

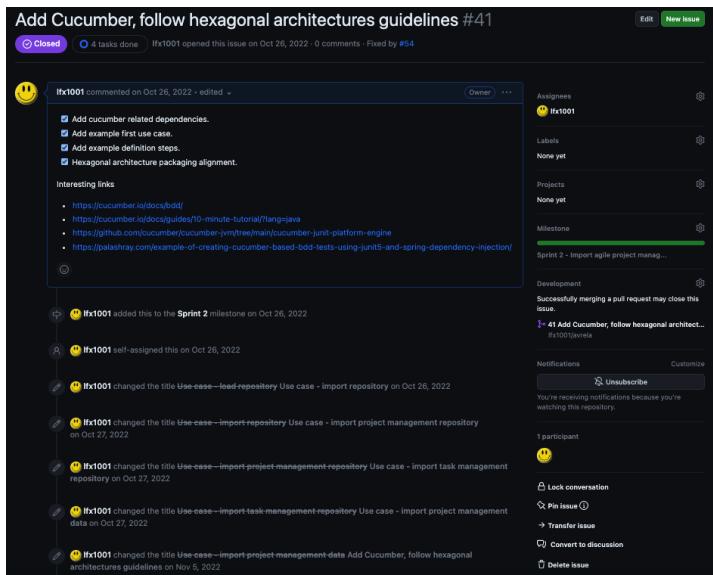


Figura 4.2: Detalle de la tarea de inclusión de Cucumber en el proyecto

A la conclusión de la mayoría de los sprints se ha realizado una revisión conjunta con el tutor a modo de retrospectiva para recabar feedback, refinar la definición de alguna de las tareas y planificar el alcance del siguiente sprint.

Para la implementación de las funcionalidades se ha realizado un desarrollo iterativo como por ejemplo para la funcionalidad correspondiente a la evaluación del laboratorio virtual de control de versiones:

- Se realiza una versión inicial con Cucumber.
- Se implementa una versión inicial con interfaz web.
- Se mejora la experiencia de usuario de la interfaz web hasta implementar la versión definitiva.

Para consolidar todos los cambios se han aplicado técnicas de integración continua y testing.

4.2. Unit testing

Unit testing (pruebas unitarias) es una técnica para la realización de pruebas de software.

Consiste en dividir el código en unidades y realizar pruebas sobre las mismas para determinar si el comportamiento es correcto [11].

En el paradigma de la Programación Orientada a Objetos, es común que estas unidades sean los métodos de una clase.

Es común que un método de una clase pueda implicar la llamada a otros métodos de la misma clase u otras clases. Para aislar el comportamiento de la unidad de código que se está probando se utilizan las siguientes técnicas:

- Mocks, fakes, stubs. La clasificación particular en una de estas categorías es inconsistente en la literatura existente [12]. En lo que sí existe acuerdo es en que:
 - Representan en el contexto de testeo al objeto o método real exponiendo la misma interfaz.
 - Devuelven respuestas predefinidas por el desarrollador.
- Test harnesses. Elementos software y de datos configurados para testear una unidad de código variando las condiciones de ejecución de la misma.

4.3. Integration testing

Consiste en combinar varias unidades de código con otros elementos software para realizar pruebas sobre el conjunto de ambos [13].

4.4. JUnit

JUnit² es un framework de pruebas que simplifica la realización de tests en Java. Con JUnit podemos desarrollar casos de test, definir asertos y ejecutar tests para verificar el comportamiento del código.

4.5. Test-driven development (TDD)

Test-driven development (TDD) - desarrollo dirigido por tests, es una técnica de desarrollo de software que consiste en convertir los requerimientos en casos de test antes de que el software esté desarrollado, en oposición a desarrollar en primer lugar el software para después realizar los tests. La creación de esta técnica se atribuye a *Kent Beck* [14].

²JUnit site: <https://junit.org/>

Esta técnica promueve un desarrollo de software incremental en el que se siguen los siguientes pasos [15]:

1. Añade un test para verificar el comportamiento de la nueva funcionalidad a desarrollar.
2. Ejecuta todos los tests. El nuevo test debe fallar ya que carece de implementación
3. Escribe una implementación sencilla para pasar el test.
4. Ejecuta todos los tests. Los tests deben pasar. De lo contrario, es necesario revisar y corregir el código desarrollado en el paso anterior.
5. Refactoriza y vuelve a ejecutar los tests después de cada refinamiento de la implementación inicial de la funcionalidad.
6. Repite otra vez estos pasos añadiendo un nuevo test.

La Figura 4.3 [16] muestra el ciclo de desarrollo TDD.

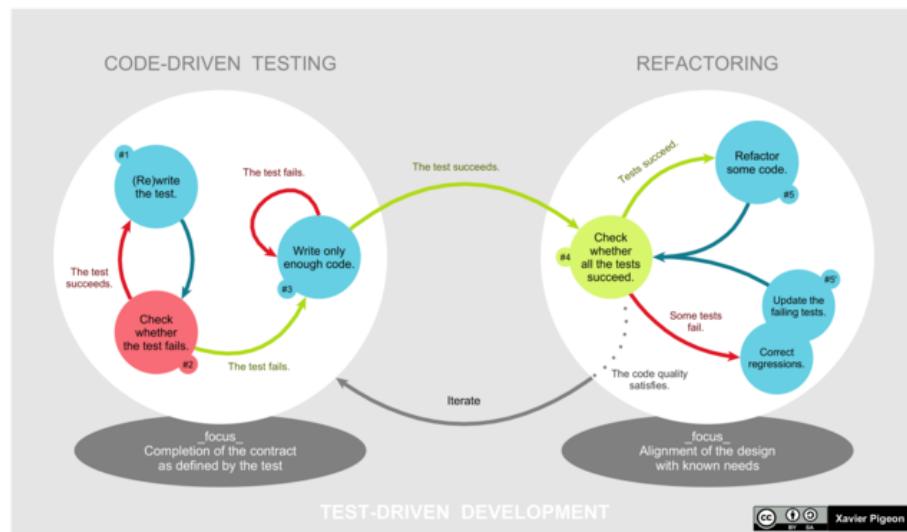


Figura 4.3: Ciclo de desarrollo TDD

4.6. User acceptance testing (UAT), Acceptance Test-Driven Development (ATDD)

En el contexto las pruebas de software, **ISTQB**³ define UAT como las pruebas realizadas desde la perspectiva de las necesidades del usuario, requisitos y procesos de negocio que determinan si un sistema satisface los criterios de aceptación en base a los cuales el usuario, cliente u otra entidad autorizada aceptará el sistema [17].

ATDD es una extensión de TDD 4.5 en la que los tests se orientan a las pruebas de aceptación a realizar.

4.7. Behaviour driven development (BDD)

BDD surge como una extensión de TDD. Es un proceso diseñado para facilitar la gestión y entrega de software mejorando la comunicación entre técnicos y expertos en el negocio [18].

BDD se basa en la idea de que ninguna persona o área tiene la respuesta para todo. Es necesaria la participación de "los tres amigos":

- Expertos en el negocio: aportan la visión de las necesidades de negocio y de los requisitos del cliente.
- Desarrolladores: estudian el impacto de implementar la funcionalidad.
- Testers: identifican obstáculos y factores adicionales a tener en cuenta de forma intuitiva, así como casos límite que puedan ocasionar problemas a corto plazo.

De una colaboración intensiva de estas áreas se obtendrán soluciones mejores, más simples y que aportarán más valor.

Siguiendo BDD especificaremos los tests poniendo el foco en el comportamiento esperado. Para facilitar la colaboración los tests se escriben en un lenguaje lo más cercano al lenguaje natural que sea posible.

La Figura 4.4 [19] muestra el ciclo de vida del desarrollo BDD.

³International Software Testing Qualifications Board



Figura 4.4: Ciclo de vida BDD

La Figura 4.5 [18] muestra la relación de los ciclos de desarrollo BDD y TDD.

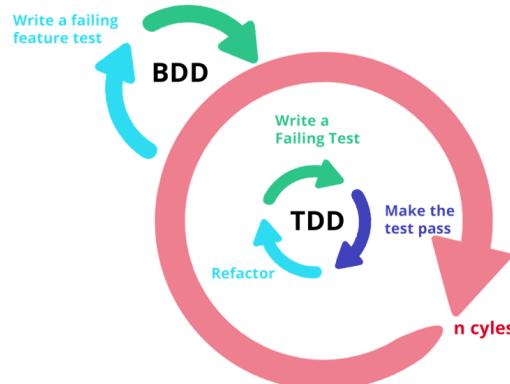


Figura 4.5: Relación de los ciclos de vida de desarrollo BDD y TDD

4.8. Cucumber

Cucumber es una herramienta software que ayuda a probar y validar la funcionalidad de las aplicaciones [20]. Se utiliza comúnmente en proyectos de desarrollo software que siguen los principios de Behavior-Driven Development (BDD) 4.7.

Cucumber permite que los desarrolladores, testers y expertos en el dominio funcional colaboren y definan el comportamiento esperado de una

aplicación de software utilizando texto en inglés. Para escribir este texto se utiliza el lenguaje estructurado Gherkin.

Feature: Import sprint data

Sprint data should be imported when input data is valid.
Uses <https://github.com/davidmigloz/go-bees> as example data.

```
Scenario Outline: Sprint import successful
  Given a repository owned by "<repoOwner>" named "<repoName>"
  And the sprint dates <sprintStartAt> and <sprintEndAt>
  When I import the sprint issues
  Then total issues should be <issues>
  And total issues labeled as documentation should be <documentation>
  And total issues labeled as feature should be <feature>
  And total issues labeled as testing should be <testing>
  And total issues labeled as bug should be <bug>
  And total issues with comments should be <withComments>
  And total issues with task list should be <withTaskList>
  And total closed issues should be <closed>
```

Examples :

	repoOwner	repoName	sprintStartAt	sprintEndAt		
	issues	documentation	feature	testing	bug	
	withComments	withTaskList	closed			
Z	davidmigloz	go-bees	2017-01-25T00:00:00Z	2017-01-25T23:59:59		
Z	17	7	4	2	4	2
Z	0	17				
Z	davidmigloz	go-bees	2017-02-01T07:00:00Z	2017-02-02T08:00:00		
Z	9	2	1	1	5	3
Z	1	9				
Z	davidmigloz	go-bees	2017-02-07T07:00:00Z	2017-02-08T08:00:00		
Z	7	4	0	1	2	1
Z	2	7				
Z	davidmigloz	go-bees	2017-02-15T07:00:00Z	2017-02-16T08:00:00		
Z	5	4	1	0	0	3
Z	1	5				

Fragmento de código 4.1: Ejemplo de sintaxis Gherkin

Con Cucumber, podemos elaborar especificaciones ejecutables conocidas como archivos de features. Estos archivos de features describen diferentes escenarios o casos de uso de la aplicación, especificando entradas, salidas esperadas y los pasos necesarios para validar el comportamiento. Estos activos se engloban en los activos de test de este proyecto como puede observarse en [4.6](#).

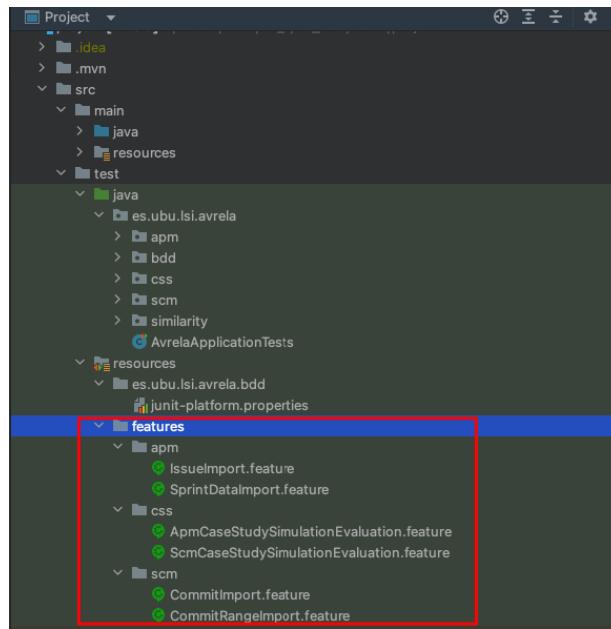


Figura 4.6: Ficheros de features de este trabajo

Cucumber actúa como un puente entre los archivos de features escritos en Gherkin y el código base subyacente de la aplicación. Analiza los archivos de features y genera pruebas ejecutables. Luego, los desarrolladores deben escribir definiciones de pasos, que son fragmentos de código que definen la implementación real de cada paso mencionado en los archivos de features. El fragmento de código 4.2. corresponde a un extracto de la definición de los pasos de la funcionalidad de importación de tareas de un sprint.

```

@Given("a repository owned by {string} named {string}")
public void aRepositoryOwnedByName(String repositoryOwner, String
repositoryName) {
    this.repositoryOwner = repositoryOwner;
    this.repositoryName = repositoryName;
}

@And("the sprint dates {zoneddatetime} and {zoneddatetime}")
public void theDatesAnd(ZonedDateTime beginAt, ZonedDateTime endAt) {
    this.beginAt = beginAt;
    this.endAt = endAt;
}

@When("I import the sprint issues")
public void iTryToImportTheRepository() {
    //Init GitHubClient
    GitHubApmClient gitHubApmClient = GitHubApmClient.with(Level.BASIC);
    GitHubMilestoneMapper milestoneMapper = GitHubMilestoneMapper.build();
    sprintRepository = new GitHubSprintRepository(gitHubApmClient,
    milestoneMapper);
    //Fetch
}

```

```

var sprints = this.sprintRepository.findByDueOnBetween(this.
repositoryOwner, this.repositoryName, this.beginAt, this.endAt);
assertNotNull(sprints);
assertEquals(1, sprints.size());
sprintUnderTest = sprints.get(0);
}
.....

```

Fragmento de código 4.2: Extracto de definición de pasos de una feature

4.9. Arquitectura hexagonal

La arquitectura hexagonal, también conocida como arquitectura de puertos y adaptadores, es un patrón de arquitectura de software que tiene como objetivo crear sistemas modulares y flexibles. Enfatiza la separación de responsabilidades y aísla la lógica de negocio de las dependencias externas, como bases de datos, interfaces de usuario y sistemas externos. Este concepto fue introducido por primera vez por Alistair Cockburn en 2005 [4].

En esta arquitectura, el sistema, a nivel lógico, se divide en tres capas. La Figura 4.7 [21] muestra de forma gráfica las distintas capas promovidas por la arquitectura hexagonal.

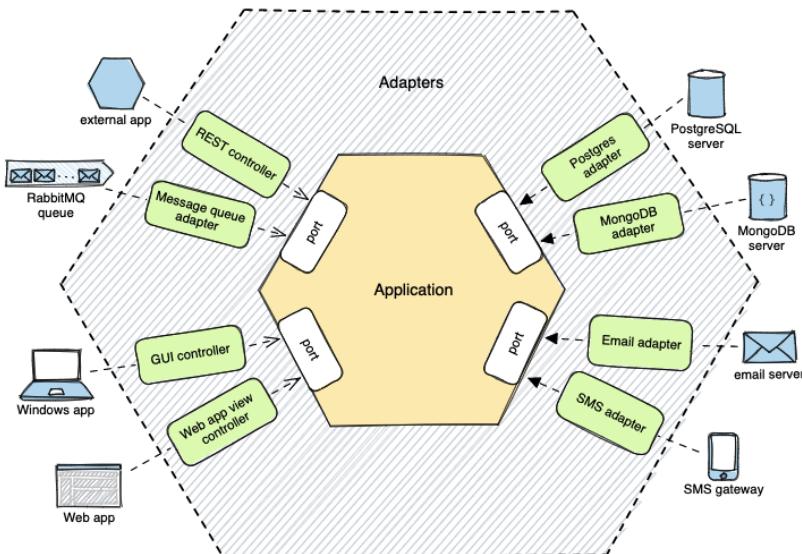


Figura 4.7: Capas de la arquitectura hexagonal

- Core o núcleo: esta es la capa más interna que contiene las reglas de negocio, el modelo de dominio y la lógica específica de la aplicación.

Representa el corazón del sistema y es responsable de las principales funcionalidades.

- Puertos (ports): Los puertos definen las interfaces o contratos a través de los cuales el core interactúa con el mundo exterior. Estos pueden ser en forma de interfaces o clases abstractas. El core define las operaciones o servicios necesarios, y los adaptadores implementan los puertos.
- Adaptadores (adapters): los adaptadores son las capas externas que proporcionan las implementaciones para los puertos. Manejan la comunicación entre el core y los sistemas externos, como bases de datos o interfaces de usuario. Los adaptadores convierten los datos recibidos de los sistemas externos a un formato que el core puede entender y viceversa.

Un sistema alineado con la arquitectura hexagonal establece lógica de negocio como núcleo y la rodea con capas de adaptadores que manejen las interacciones con sistemas externos. La lógica de negocio que conforma el núcleo es independiente de cualquier tecnología o framework específico y está diseñada para facilitar las pruebas, reemplazo y reutilización de sus componentes.

4.10. Integración continua. GitHub Actions

La integración continua es una práctica de la ingeniería de software que consiste en la inclusión de cambios de los desarrolladores en una rama principal que, ante la detección de la inclusión de cambios, lanza una serie de verificaciones para asegurar la calidad del mismo. Algunas de las verificaciones usuales son la ejecución correcta de tests y el cumplimiento de estándares de calidad mediante herramientas de análisis automático.

En este trabajo se ha implementado la integración continua mediante **GitHub Actions**⁴. GitHub actions permite la declaración de un flujo de acciones a ejecutar cuando se detectan ciertos eventos. Si se detecta algún error seremos notificados de inmediato.

La captura 4.8 corresponde al flujo de acciones a realizar ante la detección de cambios en la rama main.

⁴GitHub Actions site: <https://github.com/features/actions>

The screenshot shows a GitHub repository named 'avrela' with a file path of 'github/workflows/ci-pipeline.yml'. The code editor highlights two sections: 'EVENTOS' (Events) and 'ACCIONES' (Actions). The 'EVENTOS' section contains a 'push' event trigger with a 'branches' constraint of 'main'. The 'ACCIONES' section defines a 'build' job using an Ubuntu 20.04 runner. It includes several actions: checkout, setup Java, cache SonarCloud packages, and cache Maven. It also uses actions/cache@v1 and actions/checkout@v3. The 'ACCIONES' section ends with a command to run a Maven build.

```

name: avrela ci pipeline
on:
  push:
    branches: [ main ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK
        uses: actions/setup-java@v3
        with:
          java-version: '17'
          distribution: 'adopt'
          cache: maven
      - name: Cache SonarCloud packages
        uses: actions/cache@v1
        with:
          key: ${{ runner.os }}-sonar
          restore-keys: ${{ runner.os }}-sonar
      - name: Cache Maven packages
        uses: actions/cache@v2
        with:
          key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
          restore-keys: ${{ runner.os }}-m2
      - name: Build and analyze
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }} # Needed to get PR information, if any
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
        run: mvn -B -f project/pom.xml verify org.sonarsource.scanner.maven:sonar-scanner-project:1.0.1801 -DSONAR_TOKEN=$SONAR_TOKEN

```

Figura 4.8: Definición de acciones de integración continua en GitHub actions

Utilizando la herramienta Maven 4.13 por línea de comandos se realiza el siguiente flujo de acciones:

- Compilación de código.
- Ejecución de test unitarios, de integración y tests realizados con Cucumber 4.8.
- Análisis de calidad con SonarCloud 4.17.

Desde GitHub podemos acceder al **histórico del flujo de acciones de integración continua**⁵ de integración continua.

⁵ Histórico de ejecuciones: <https://github.com/lfx1001/avrela/actions>

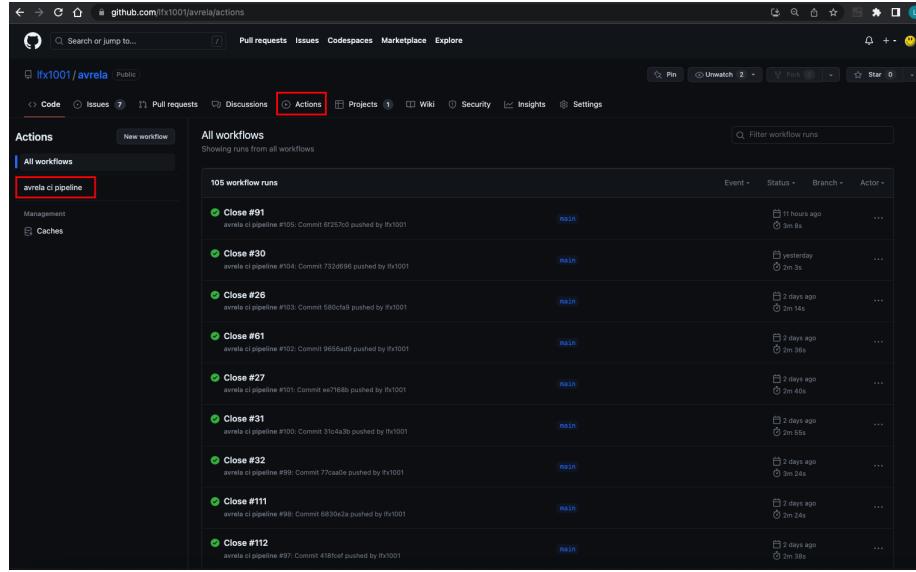


Figura 4.9: Histórico de ejecuciones de flujos de acción

4.11. Insomnia

Insomnia⁶ es una aplicación cliente de servicios web que facilita la depuración, testeo y exploración de APIs REST y GraphQL.

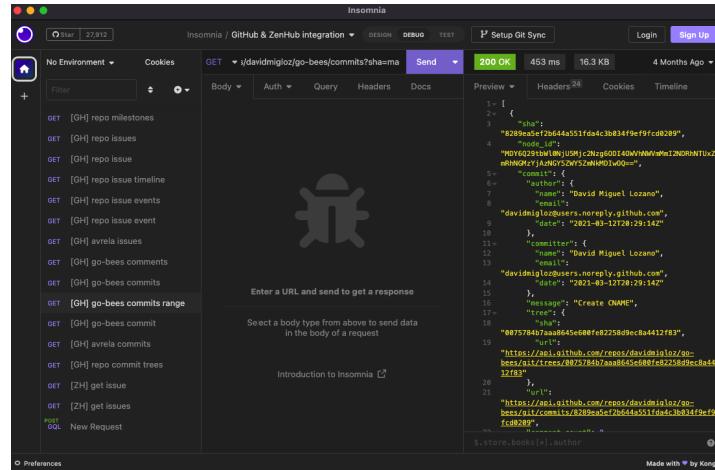


Figura 4.10: Captura de la herramienta Insomnia

⁶Insomnia site: <https://insomnia.rest/>

En el contexto de este trabajo Insomnia se ha utilizado para explorar y probar el API de GitHub y Zenhub.

4.12. Open Feign

Open Feign⁷ es una librería que simplifica la implementación de clientes de servicios web.

Proporciona un modelo declarativo que permite definir clientes de servicios web añadiendo una serie de anotaciones a métodos de una interfaz Java. En tiempo de ejecución, a partir de estas anotaciones en los métodos, se genera el cliente java efectivo.

En este trabajo se ha utilizado Open Feign para la interacción con el API de GitHub. Por ejemplo, para la operación de obtención de información de un commit, se muestra en [4.11](#) la implementación en Open Feign.

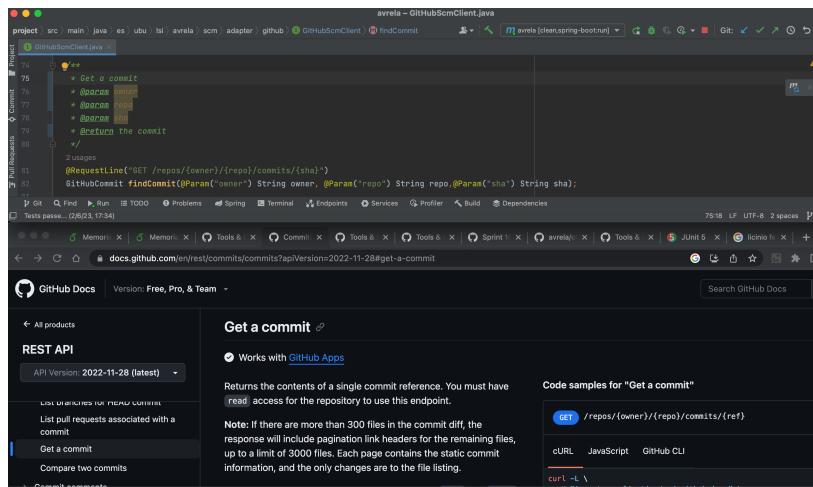


Figura 4.11: Implementación de la operación de obtención de un commit utilizando Open Feign

4.13. Maven

Maven⁸ es una herramienta para la gestión de los activos de código que simplifica la construcción, empaquetado y gestión de dependencias de un proyecto.

⁷Open Feign site: <https://github.com/OpenFeign/feign>

⁸Maven site: <https://maven.apache.org/>

Entre sus características destacan:

- Estructura estándar de proyectos.
- pom.xml, más conocido como Project Object Model. Contiene la configuración de un proyecto como el nombre del proyecto, versión, dependencias y plugins utilizados.
- Gestión de dependencias: En el pom.xml se especifican las librerías y versiones de las mismas utilizadas. Maven se encarga de descargar las mismas desde repositorios de artefactos públicos o privados y hacerlas accesible a nuestro proyecto.
- Automatización de la construcción: Maven define una serie de fases estándar como compilación, ejecución de tests, empaquetado de forma que con un solo comando podemos realizar todas las acciones requeridas para la construcción.
- Extensibilidad: permite el uso de plugins adicionales así como el desarrollo de plugins propios.

4.14. Spring Boot

Spring Boot⁹ es un framework de desarrollo de aplicaciones Java.

Su objetivo es mejorar la productividad de los desarrolladores proporcionando una plataforma en la que gran parte de los componentes se pre-configuran y están listos para ser utilizados de forma transparente para el desarrollador. Se enumeran a continuación los aspectos clave en los que se basa Spring Boot:

- **Configuración simplificada:** En lugar de complejos ficheros XML de configuración esta se realiza con ajustes mínimos basados en convenciones.
- **Servidor embebido:** Incluye servidores web embebidos como Tomcat o Jetty permitiendo que las aplicaciones sean autocontenidoas y puedan ejecutarse como un fichero JAR convencional.

⁹Spring Boot site: <https://spring.io/projects/spring-boot>

- **Auto configuración:** Analiza las dependencias del proyecto y en base a las dependencias preconfigura siguiendo su propio estándar servicios asociados, como por ejemplo, el servidor web en el puerto 8080.
- **Gestión de dependencias:** Spring facilita la gestión de dependencias mediante el uso de starters. Estos starters son conjuntos de librerías para proporcionar una capacidades técnicas determinadas cuyas versiones son compatibles entre sí. Entre otros muchos, existen starters para aplicaciones web, acceso a base de datos y seguridad.
- **Integración con el ecosistema Spring:** Spring Boot ha sido concebido para integrarse de forma natural con otros componentes del ecosistema Spring como Spring Data, Spring Security y Spring MVC.

4.15. Thymeleaf

Thymeleaf¹⁰ es un motor de plantillas desarrollado en Java utilizado para generar, entre otros, HTML dinámico, por lo que se utiliza en desarrollos Java web como alternativa al uso de la tecnología JSP o Freemarker. Thymeleaf dispone de integración con Spring por lo que los ajustes de configuración para su uso en este trabajo han sido mínimos.

4.16. Bootstrap

Bootstrap¹¹ es un framework para realizar páginas webs atractivas y responsive mediante un conjunto de plantillas, estilos y componentes predefinidos utilizando HTML, CSS y JavaScript.

En este trabajo se hace uso de Bootstrap para proporcionar una experiencia de usuario mejor en su interacción con la interfaz web. Un ejemplo de uso se incluye en [4.12](#), en el que mediante el componente navbar se define la barra de navegación superior con el logo del proyecto y el escudo de la universidad.

¹⁰ Thymeleaf site: <https://www.thymeleaf.org/>

¹¹ Bootstrap site: <https://getbootstrap.com/>

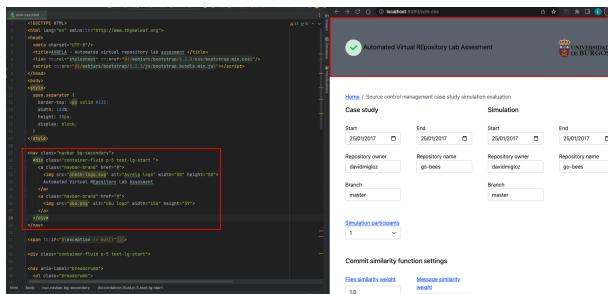


Figura 4.12: Uso de componente bootstrap navbar y visualización

4.17. Sonarcloud

SonarCloud¹² es un servicio online que ayuda a los desarrolladores a mejorar la calidad de su código analizándolo en busca de problemas, errores, code smells y vulnerabilidades. Esto proporciona una revisión de código automatizada para guiar a los desarrolladores a mantener un código limpio, eficiente y seguro.

En este trabajo se ha configurado el análisis con SonarCloud como un paso de la integración continua 4.10. El análisis SonarCloud de este trabajo está accesible en esta url¹³.

4.18. Chat GPT

Para implementar el algoritmo de la distancia de Jaro-Winkler 3.6, en el contexto de este trabajo, se plantean varias alternativas:

- Implementar el algoritmo desde cero a partir de su definición formal.
- Reutilizar una implementación del algoritmo ya existente en alguna librería o fuente de confianza.
- Probar la funcionalidad ofrecida por la plataforma Chat GPT.

Finalmente, se escoge la opción de Chat GPT con el objetivo de experimentar con esta plataforma.

¹²SonarCloud site: <https://www.sonarsource.com/products/sonarcloud/>

¹³SonarCloud site de este trabajo: https://sonarcloud.io/project/overview?id=lfx1001_avrela

Para ello, una vez hemos accedido a la plataforma, indicamos que queremos obtener una implementación Java del algoritmo de Jaro-Winkler:

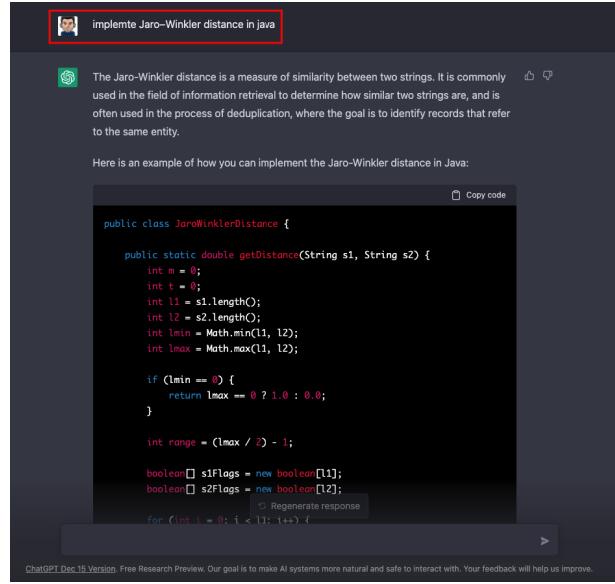


Figura 4.13: Petición de implementación de algoritmo a Chat GPT

A continuación, solicitamos a chat GPT que proporcione la implementación de tests unitarios para la implementación Java del algoritmo de Jaro-Winkler:

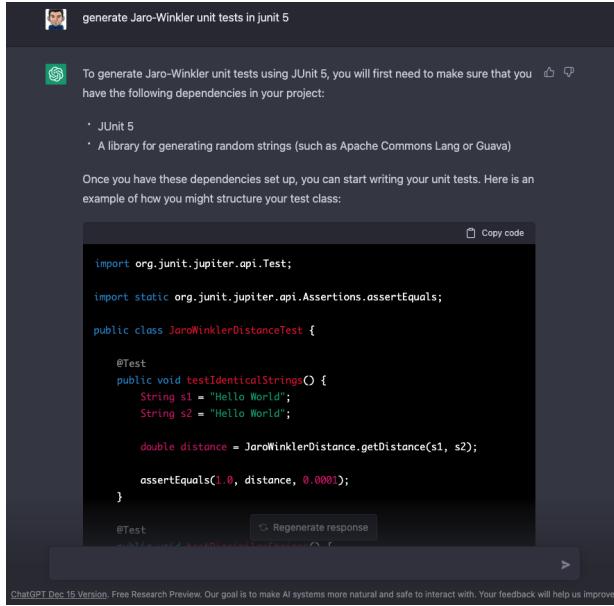


Figura 4.14: Petición de implementación de tests unitarios para la implementación Java del algoritmo de Jaro-Winckler

Las fragmentos de código obtenidos en ambos casos han compilado sin modificación alguna.

En cuanto a la calidad del código generado, destacar que el nombrado de variables del código obtenido no es intuitivo y que el análisis mediante SonarCloud arroja perspectivas de mejora en cuanto a la calidad del mismo, como por ejemplo a nivel de complejidad ciclomática:

The screenshot shows a SonarCloud analysis report for the file `JaroWinklerDistance.java`. It displays three issues:

- Add a private constructor to hide the implicit public one.** (Code Smell, Open, Major, Not assigned) - 5min effort, 4 months ago
- Refactor this method to reduce its Cognitive Complexity from 27 to the 15 allowed.** (Code Smell, Open, Critical, Not assigned) - 17min effort, 4 months ago
- Reduce the total number of break and continue statements in this loop to use at most one.** (Code Smell, Open, Minor, Not assigned) - 20min effort, 4 months ago

1 / 3 issues 42min effort

src.../es/ubu/is/avrela/similarity/JaroWinklerDistance.java

3 of 3 shown

Figura 4.15: Análisis Sonar Cloud de la implementación Java del algoritmo de Jaro-Winkler

4.19. Lombok

Lombok¹⁴ es una librería Java que se utiliza para reducir el código fuente repetitivo.

Algunos ejemplos de código repetitivo del lenguaje Java son:

- Accessor methods (Getters y Setters).
- Constructores.
- Método `toString`.
- Métodos `equals` y `hashCode`.

Mediante el uso de anotaciones, Lombok genera el código repetitivo en tiempo de compilación en los ficheros pseudocompilados ".class".

Por ejemplo, una clase para representar un empleado, de forma convencional, se codificaría según el fragmento de código 4.3.

```
public class Employee {
    private Integer employeeId;
    private String name;
    private String company;
    private String emailId;

    public Employee() {}

    public Employee(Integer employeeId, String name,
                   String company, String emailId)
    {
        super();
        this.employeeId = employeeId;
        this.name = name;
        this.company = company;
        this.emailId = emailId;
    }

    public Integer getEmployeeId() { return employeeId; }

    public void setEmployeeId(Integer employeeId)
    {
        this.employeeId = employeeId;
    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getCompany() { return company; }
```

¹⁴Project Lombok: <https://projectlombok.org/>

```

public void setCompany( String company )
{
    this.company = company;
}

public String getEmailId() { return emailId; }

public void setEmailId( String emailId )
{
    this.emailId = emailId;
}

@Override public String toString()
{
    return "Employee ["
        + "employeeId=" + employeeId + ", name=" + name
        + ", "
        + "company=" + company + ", emailId=" + emailId
        + "]";
}
}

```

Fragmento de código 4.3: Clase Java convencional

Con lombok, el código fuente quedará como el fragmento de código 4.4.

```

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@NoArgsConstructor
@AllArgsConstructor
@ToString
public class Employee {
    private @Getter @Setter Integer employeeId;
    private @Getter @Setter String name;
    private @Getter @Setter String company;
    private @Getter @Setter String emailId;
}

```

Fragmento de código 4.4: Clase Java utilizando Lombok

Esta disminución de código repetitivo redonda en una mayor legibilidad del código fuente, favoreciendo su mantenimiento.

Aspectos relevantes del desarrollo del proyecto

5.1. Integración con GitHub

La información correspondiente a la planificación de proyectos ágiles y control de versiones reside en GitHub.

Elección de API para la interacción con GitHub

Para la interacción con GitHub existen dos tipos de APIs a disposición de los desarrolladores:

- API REST.
- API GraphQL.

De entre ambas opciones, nos decantamos por el API REST ya que, si bien el API GraphQL ofrece una gran flexibilidad a la hora de personalizar las consultas para agregar recursos y granular los datos recuperados con la consiguiente reducción de llamadas, el API REST ofrece un mayor ecosistema de técnicas y herramientas que facilitarán la integración.

Validamos en primer lugar la viabilidad de utilizar el API REST utilizando el cliente de servicios web Insomnia [4.11](#). En base a la documentación del API REST de GitHub realizamos una serie de pruebas de acceso a los recursos necesarios:

- Milestones.

- Issues.
- Comments.

Constatamos de esta forma que todos los datos correspondientes a la gestión de tareas necesarios están disponibles.

En el consumo del API REST de GitHub hemos de considerar que aplican **restricciones en el número de peticiones¹⁵**:

- Peticiones autenticadas: 5000 peticiones por usuario y hora.
- Peticiones sin autenticar: 60 peticiones por dirección IP y hora.

Desarrollo de un cliente GitHub a medida

El consumo del API REST de GitHub puede llevarse a cabo mediante:

- Uso de librerías cliente ya existentes, como por ejemplo **GitHub API for Java¹⁶**
- Desarrollo de un cliente a medida.

Dado que el conjunto de operaciones del API de GitHub necesario para el desarrollo de este trabajo es reducido se opta por desarrollar un cliente a medida mediante Open Feign [4.12](#).

Pruebas de integración del cliente GitHub a medida

Para verificar el cliente GitHub, planteamos una serie de tests de integración [4.3](#) mediante Cucumber [4.8](#), reflejados en la Figura [5.1](#) para comprobar que la información capturada de forma automática coincide con la información capturada de forma manual en el enunciado del caso de estudio de simulación.

¹⁵GitHub API limit rates: <https://docs.github.com/en/rest/overview/resources-in-the-rest-api>

¹⁶GitHub API for Java: <https://github-api.kohsuke.org/>

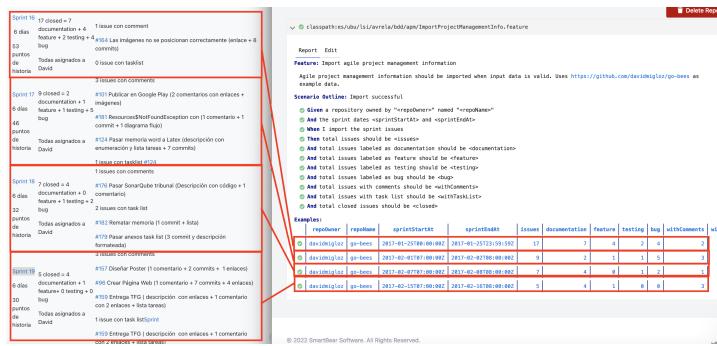


Figura 5.1: Comparativa de la información de gestión de tareas obtenida de forma manual y automática

5.2. Integración con Zenhub

ZenHub es la plataforma en la que reside la información de los puntos de historia de usuario de las tareas. Procedemos de forma análoga al estudio realizado para la interacción con GitHub, realizando peticiones contra el API REST de ZenHub.

Descubrimos que la interacción con proyectos en ZenHub requiere de la **generación de un token de autenticación¹⁷** por parte del propietario del proyecto . En el contexto del estudio de casos de simulación, cada estudiante debería generar dicho token para que el evaluador pueda acceder a la información.

Esto plantea una dificultad añadida para la evaluación de los casos de estudio de simulación, a diferencia de la interacción con GitHub, que no requiere de acción alguna para extraer información de los proyectos siempre que el repositorio sea público.

Tras la finalización del Sprint 1 decidimos posponer la **tarea de integración con Zenhub¹⁸** y, finalmente, se acuerda con el tutor eliminar del alcance de este trabajo la integración de Zenhub y con ello la información relativa a las estimaciones.

¹⁷Autenticación de llamadas al API de ZenHub: <https://github.com/ZenHubIO/API#authentication>

¹⁸Tarea de integración con Zenhub: <https://github.com/lfx1001/avrela/issues/10>

5.3. Comunicación entre alumno y tutor mediante el flujo de trabajo con GitHub

En el repositorio GitHub de este proyecto el tutor ha sido dado de alta como colaborador. Los colaboradores pueden configurar una serie de notificaciones para los eventos que suceden en el repositorio.

En el caso tareas simples, que puedan realizarse en un solo commit, se ha referenciado la tarea en el mensaje del commit [5.2](#). Esto desencadena un evento de cierre de tarea que es notificado al tutor.

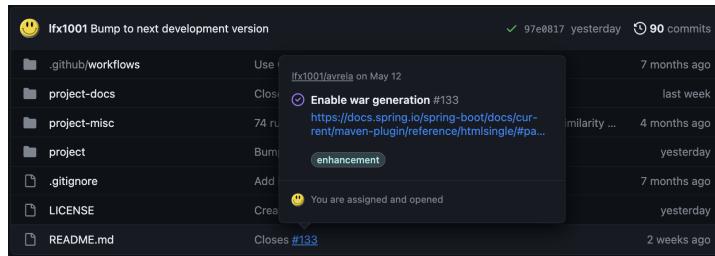


Figura 5.2: Captura cierre de tarea referenciada en el commit

En el caso de tareas complejas, se ha creado una rama a partir de la tarea mediante GitHub [5.3](#).

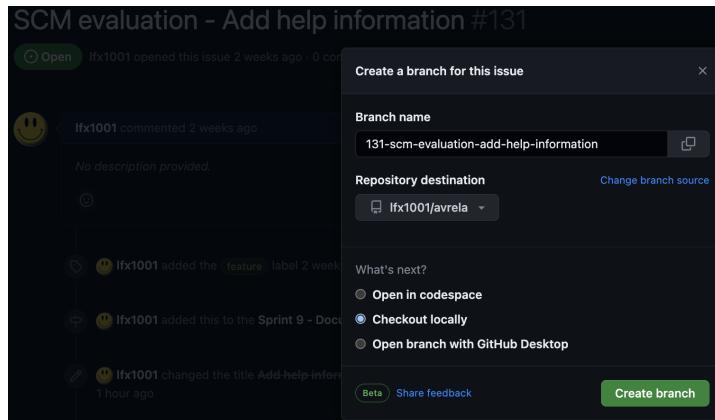


Figura 5.3: Creación de rama a partir de una tarea

El alumno procede a trabajar sobre la nueva rama [5.4](#) en su repositorio local sincronizando periodicamente sus cambios locales con el repositorio remoto.

5.3. COMUNICACIÓN ENTRE ALUMNO Y TUTOR MEDIANTE EL FLUJO DE TRABAJO CON GITHUB

41

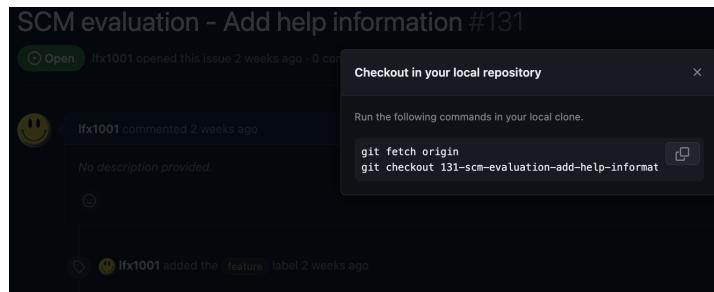


Figura 5.4: Instrucciones para descargar la rama asociada a la tarea en el repositorio local

La existencia de cambios en ramas de los repositorios es notificada en la propia interfaz web de GitHub [5.5](#).

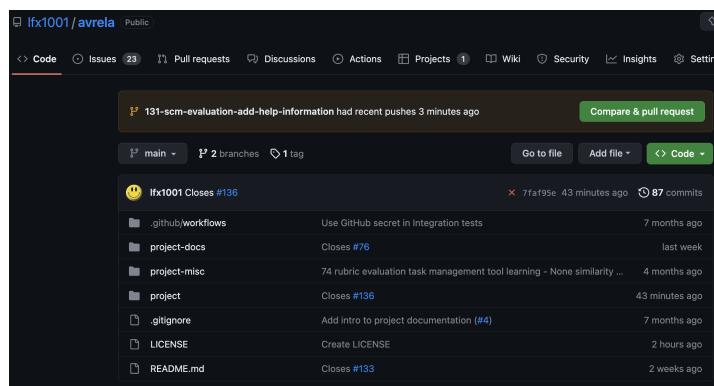


Figura 5.5: Notificación de cambios recientes en una rama

Desde la pantalla anterior se permite revisar los cambios así como crear una pull request. Cuando se realiza el merge de esta pull request GitHub dará por resultado la tarea asociada [5.6](#), notificando el evento de cierre de tarea al tutor que podrá revisar la misma gracias a la trazabilidad entre la tarea y la pull request.

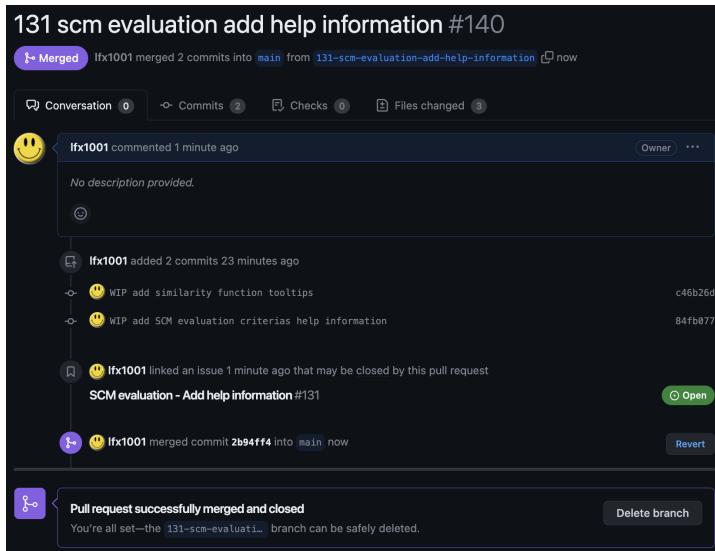


Figura 5.6: Detalle de pull request mergeada y cerrada

5.4. Trazabilidad en GitHub de la aplicación de BDD y TDD

Para implementar algunas de las funcionalidades de este trabajo se ha hecho uso de las técnicas de TDD [4.5](#) y BDD [4.7](#).

La aplicación de dichas técnicas puede contrastarse por ejemplo en la **pull request asociada a la tarea de evaluar la trazabilidad de commits con la herramienta de gestión de proyectos**¹⁹

En el detalle de la pull request podemos visualizar los commits parciales y su relación con etapas asociadas de las técnicas de TDD [4.5](#) y BDD [4.7](#) [5.7](#).

¹⁹98 scm simulation commit apm issue reference evaluation pull request: <https://github.com/lfx1001/avrela/pull/103>

5.4. TRAZABILIDAD EN GITHUB DE LA APLICACIÓN DE BDD Y TDD

43

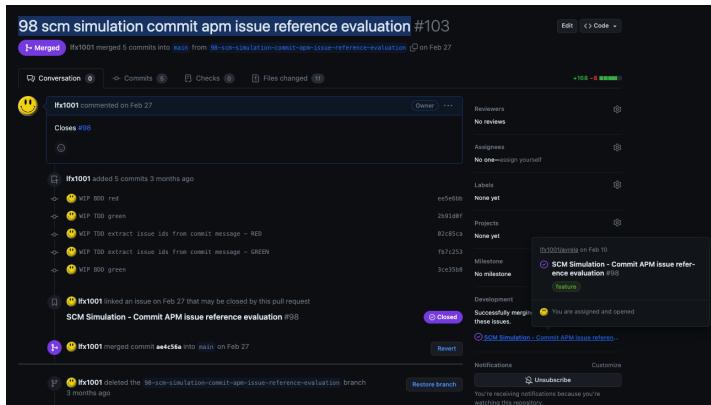


Figura 5.7: Detalle de la pull request en GitHub

Desde el detalle de la pull request podemos visualizar los cambios asociados a cada uno de los commits [5.8](#).

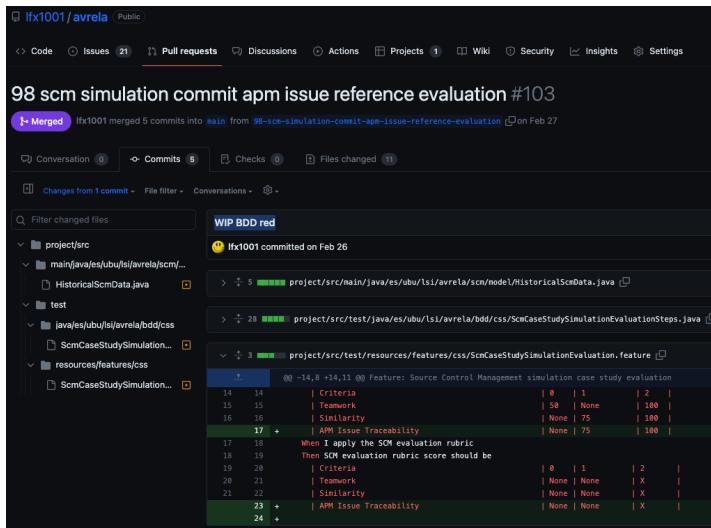


Figura 5.8: Detalle del commit con mensaje WIP BDD red

5.5. Interpretación de resultados obtenidos en la evaluación del laboratorio virtual de control de versiones

En las pruebas de evaluación del laboratorio virtual de control de versiones observamos como en el caso de estudio aparecen en algunos casos commits que no estaban contemplados en las instrucciones del laboratorio virtual y que estos provocan una desalineamiento de los commits.

Por ejemplo, en la evaluación del laboratorio virtual de control de versiones con los datos de entrada definidos en 5.9:

The screenshot shows a web-based simulation interface for evaluating a source control management case study. It includes fields for 'Case study' (Start: 16/02/2017, End: 02/01/2017), 'Simulation' (Start: 09/12/2022, End: 09/12/2022), repository details (Repository owner: davidmiglez, Repository name: go-bees, Repository owner: pac1006, Repository name: GESPRO_GESTIONVERSIONES), and a branch selection (Branch: master). Under 'Commit similarity function settings', there are fields for 'Files similarity weight' (1.0) and 'Message similarity weight' (1.0). A 'Similarity threshold' field is set to 0.75. A 'Submit' button is at the bottom.

Figura 5.9: Datos de entrada ejemplo para la evaluación del laboratorio virtual de control de versiones

se observa como el commit N del caso de estudio se corresponde con el commit N-1 de la simulación como puede observarse en 5.10

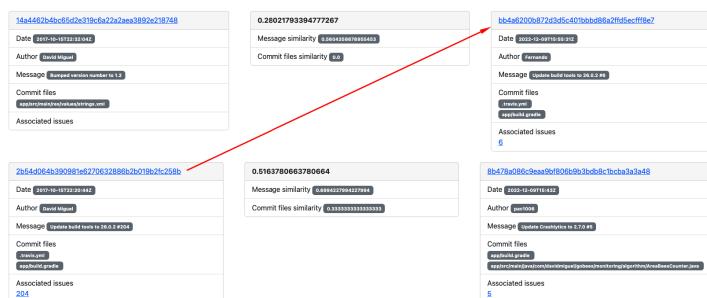


Figura 5.10: Desalineamiento de commits

5.6. DISCREPANCIAS ENTRE LA RÚBRICA DE LA EVALUACIÓN AUTOMÁTICA Y LA UTILIZADA EN LA EVALUACIÓN MANUAL 45

El commit del caso de estudio responsable del desalineamiento es un merge commit, por lo tanto, para mitigar este comportamiento se podrían implementar alguna de estas soluciones:

- excluir los merge commits de la importación.
- permitir especificar casos de estudio y simulaciones mediante un commit inicial y un commit final.

5.6. Discrepancias entre la rúbrica de la evaluación automática y la utilizada en la evaluación manual

En el desarrollo de este trabajo se ha planteado rúbricas de evaluación alternativas a las originales.

En el caso de la rúbrica de evaluación automática del laboratorio de gestión de proyectos ágil, se han eliminado dos de los criterios de evaluación dado que se ha decidido eliminar del alzance del trabajo la importación de estimaciones temporales de Zenhub [?]:

- estimaciones temporales.
- planificación de sprint.

Para la rúbrica de evaluación automática del laboratorio de control de versiones, el criterio de simulación de control de versiones se ha valorado sobre la misma escala de valores que el resto de criterios de evaluación (0,1,2), en lugar de la escala (0,2,4,6).

5.7. Distribución y revisión de la aplicación

Este trabajo se ha desarrollado utilizando tecnología Java principalmente. A diferencia de los desarrollos de aplicaciones web Java convencionales, en los que el código y activos relacionados se empaqueta en un artefacto .war o .ear para su posterior despliegue en un servidor de aplicaciones, el uso de Spring Boot [4.14](#) permite generar empaquetados .jar autocontenidos que incluyen los componentes del servidor web, por lo que puede ser ejecutado en un entorno local de forma sencilla [5.11](#).

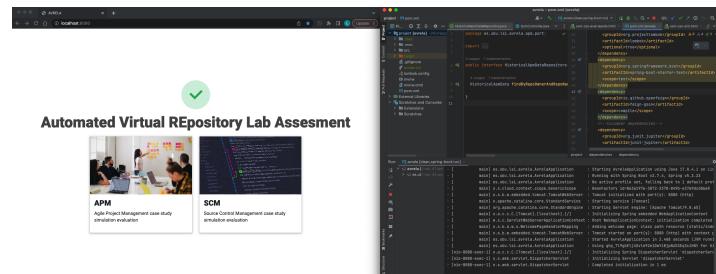


Figura 5.11: Ejecución y acceso desde un entorno local a la aplicación

De esta forma, la revisión de la aplicación por parte de alumno y tutor se simplifica.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

Una vez finalizado este trabajo, se plantean una serie de conclusiones acerca del mismo.

Podemos afirmar que, a nivel general, los objetivos funcionales del proyecto se han cumplido. Las evaluaciones de ambos laboratorios virtuales se han automatizado si bien no con un grado de fidelidad total debido a que se ha eliminado del alcance importar las estimaciones temporales de Zenhub y se han realizado modificaciones sobre las rúbricas de evaluación [5.6](#).

La interfaz web proporciona feedback acerca de las características de los elementos a considerar en los diferentes criterios de evaluación de las rúbricas. El disponer en una sola pantalla de una visión unificada de los elementos del caso de estudio y de la simulación con sus respectivos enlaces a GitHub facilita la revisión de posibles discrepancias.

Los requisitos técnicos planteados en el inicio del trabajo también se han cumplido. Se han utilizado técnicas de gestión de proyectos ágiles [4.1](#) así como técnicas de aseguramiento de calidad integrando desde la fase inicial de codificación el análisis automático con SonarCloud [4.17](#).

La adopción de la arquitectura hexagonal ha permitido desacoplar la lógica de negocio de las evaluaciones automáticas de las fuentes de información y, por tanto, basta con desarrollar nuevas implementaciones de los adaptadores para extender la funcionalidad de este trabajo.

7.2. Líneas de trabajo futuras

La conclusión de este trabajo ofrece un punto de partida para la mejora del producto. En este sentido, se plantean a continuación líneas de trabajo futuras.

La importación de casos de estudio y simulaciones de control de versiones podría realizarse especificando un commit de inicio y un commit final. De esta forma se proporciona una granularidad más fina respecto a la ofrecida actualmente basada en rango de fechas.

Las funcionalidades proporcionadas por este trabajo permiten automatizar tareas de evaluación en el ámbito académico. Estas funcionalidades podrían ampliarse y explotarse en otros ámbitos de aplicación. Se proponen a continuación dos supuestos:

- evaluación de competencias técnicas en procesos de selección personal: los candidatos podrían realizar una simulación que sería contrastada con el caso de estudio de referencia con el fin de determinar si son aptos o no en la competencia en cuestión.
- generación de planificaciones a partir de casos de estudio: la herramienta podría ingestar una planificación de proyectos ágiles de un sistema origen y copiarla a un sistema destino. La variedad de adaptadores de la herramienta determinaría el grado de interoperabilidad. Actualmente, la herramienta únicamente permite extraer información de GitHub por lo que sería necesario implementar adaptadores para otras herramientas, como **JIRA**²⁰, tanto de entrada como de salida.

Con respecto al rendimiento, debido al gran número de llamadas a GitHub, los tiempos de ejecución se incrementan en función del número de elementos que componen el caso de estudio y la simulación. Para mejorar estos tiempos, podría plantearse sustituir los adaptadores actuales del API Rest de GitHub por adaptadores para el API GraphQL de GitHub.

²⁰ JIRA: <https://www.atlassian.com/software/jira>

Bibliografía

- [1] J. Bourne, D. Harris, and F. Mayadas, “Online engineering education: Learning anywhere, anytime,” *Online Learning*, vol. 9, 03 2019.
- [2] I. Tejado, E. Hernández, I. González, P. Merchán, and B. Vinagre, “Introducing automatic evaluation in virtual laboratories for control engineering at university of extremadura. first steps,” *International Journal of Mathematics and Computers in Simulation*, vol. 12, pp. 55–63, 04 2018.
- [3] P. Sánchez, Garnacho, J. Dacosta, and E. Mandado, “El aprendizaje activo mediante la autoevaluación utilizando un laboratorio virtual.” *IEEE-RITA*, vol. 4, pp. 53–62, 01 2009.
- [4] A. Cockburn, “Hexagonal architecture,” 2005, [Internet; consultado 14-octubre-2022]. [Online]. Available: <https://alistair.cockburn.us/hexagonal-architecture/>
- [5] A. Alliance, “Agile glossary,” [Internet; consultado 14-octubre-2022]. [Online]. Available: <https://www.agilealliance.org/agile101/agile-glossary/>
- [6] Wikipedia, “Control de versiones — wikipedia, la enciclopedia libre,” 2022, [Internet; descargado 25-octubre-2022]. [Online]. Available: https://es.wikipedia.org/wiki/Control_de_versiones
- [7] Git, “gitglossary - a git glossary,” 2022, [Internet; descargado 25-octubre-2022]. [Online]. Available: <https://git-scm.com/docs/gitglossary>

- [8] Wikipedia, “Similarity measure,” 2022, [Internet; consultado 12-mayo-2023]. [Online]. Available: https://en.wikipedia.org/wiki/Similarity_measure
- [9] ——, “Jaccard index,” 2022, [Internet; consultado 12-mayo-2023]. [Online]. Available: https://en.wikipedia.org/wiki/Jaccard_index
- [10] ——, “Jaro-winkler distance,” 2022, [Internet; consultado 12-mayo-2023]. [Online]. Available: https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance
- [11] ——, “Unit testing,” 2022, [Internet; consultado 12-noviembre-2022]. [Online]. Available: https://en.wikipedia.org/wiki/Unit_testing
- [12] ——, “Mock object,” 2022, [Internet; consultado 12-noviembre-2022]. [Online]. Available: https://en.wikipedia.org/wiki/Mock_object#Mocks.2C_fakes.2C_and_stubs
- [13] ——, “Integration testing,” 2022, [Internet; consultado 12-noviembre-2022]. [Online]. Available: https://en.wikipedia.org/wiki/Integration_testing
- [14] ——, “Test-driven development,” 2022, [Internet; consultado 12-noviembre-2022]. [Online]. Available: https://en.wikipedia.org/wiki/Test-driven_development
- [15] K. Beck, *Test-Driven Development by Example*. Addison Wesley, 2002.
- [16] X. Pigeon, “Lifecycle of the test-driven development method,” 2015, [Internet; consultado 12-noviembre-2022]. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/0/0b/TDD_Global_Lifecycle.png
- [17] Wikipedia, “Acceptance testing,” 2022, [Internet; consultado 13-noviembre-2022]. [Online]. Available: https://en.wikipedia.org/wiki/Acceptance_testing
- [18] Digité, “Behavior driven development (bdd): Creating useful software customers want,” 2022, [Internet; consultado 13-noviembre-2022]. [Online]. Available: <https://www.digité.com/agile/behavior-driven-development-bdd/>
- [19] T. point, “Cucumber - overview,” 2022, [Internet; consultado 14-noviembre-2022]. [Online]. Available: https://www.tutorialspoint.com/cucumber/cucumber_overview.htm

- [20] S. Software, “Hexagonal architecture explained,” 2019, [Internet; consultado 3-junio-2023]. [Online]. Available: <https://cucumber.io/docs/guides/overview/>
- [21] A. Huttunen, “Hexagonal architecture explained,” 2023, [Internet; consultado 3-junio-2023]. [Online]. Available: <https://www.arhohuttunen.com/hexagonal-architecture/>