



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 — 10 de diciembre  
de 2022

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 dni, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

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, 10 de diciembre de 2022

Vº. Bº. del Tutor:

D. Dr. Carlos López Nozal





## Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

## Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

## **Abstract**

A **brief** presentation of the topic addressed in the project.

## **Keywords**

keywords separated by commas.



---

# Índice general

---

Índice general	iii
Índice de figuras	v
Índice de tablas	vi
Introducción	1
Objetivos del proyecto	5
Conceptos teóricos	7
3.1. Laboratorios virtuales. . . . .	7
3.2. Conceptos de la gestión de proyectos ágiles utilizados en el laboratorio virtual. . . . .	7
3.3. Conceptos de control de versiones utilizados en el laboratorio virtual . . . . .	9
Técnicas y herramientas	11
4.1. Metodología de desarrollo del proyecto . . . . .	11
4.2. Unit testing . . . . .	11
4.3. Integration testing . . . . .	12
4.4. Test-driven development (TDD) . . . . .	12
4.5. User acceptance testing (UAT), Acceptance Test-Driven Development (ATDD) . . . . .	13
4.6. Behaviour driven development (BDD) . . . . .	13
4.7. Cucumber . . . . .	15
4.8. Domain driven design (DDD) . . . . .	17

4.9. Arquitectura hexagonal . . . . .	17
4.10. Integración continua . . . . .	17
4.11. Insomnia . . . . .	17
4.12. Pipeline . . . . .	17
4.13. Open Feign . . . . .	18
4.14. Sonarqube . . . . .	18
4.15. Lombok . . . . .	18
<b>Aspectos relevantes del desarrollo del proyecto</b>	<b>21</b>
5.1. Obtención de información correspondiente a la gestión de tareas del proyecto . . . . .	21
<b>Trabajos relacionados</b>	<b>25</b>
<b>Conclusiones y Líneas de trabajo futuras</b>	<b>27</b>
<b>Bibliografía</b>	<b>29</b>

---

# 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. Ciclo de desarrollo TDD . . . . .	13
4.2. Ciclo de vida BDD . . . . .	14
4.3. Relación de los ciclos de vida de desarrollo BDD y TDD . . . . .	15
5.1. Comparativa de la información de gestión de tareas obtenida de forma manual y automática . . . . .	23

---

## Índice de tablas

---

---

# Índice de fragmentos de código

---

4.1. Gherkin sample code . . . . .	15
4.2. Clase Java convencional . . . . .	18
4.3. Clase Java utilizando Lombok . . . . .	19



---

# 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 a 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 alumno 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 alumnos y reducir la carga de trabajo de los docentes. 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):

- 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 alumno cuenta con el repositorio original a modo de referencia.

<b>Sprint 16</b>	17 closed = 7	
	documentation + 4	1 issue con comment
6 días	feature + 2 testing + 4	
	bug	<a href="#">#164</a> 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.



Valoración del trabajo en equipo en Github	Participación menos del 50% de los miembros del equipo en las tareas		Participación del 100% de los miembros del equipo en las tareas	
	0 points		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	Se aplican tiempos a todos los elementos Scrum planificados. Los tiempos se aproximan a la realidad.
			1 points	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 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. Laboratorios virtuales.

TBD

### 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.

TBD - Incluir captura historia de usuario.

#### 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:

TBD - Incluir captura sprint

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 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.

TBD - Incluir captura tarea

### 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 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.

## GitHub

Es un servicio de alojamiento en internet de repositorios Git.



---

# Técnicas y herramientas

---

## 4.1. Metodología de desarrollo del proyecto

TBD: Uso de metodologías ágiles - Inclusión de capturas de los Sprints en GitHub

## 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 [8].

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 [9]. 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 [10].

### 4.4. 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* [11].

Esta técnica promueve un desarrollo de software incremental en el que se siguen los siguientes pasos [12]:

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 4.1 [13] muestra el ciclo de desarrollo TDD.

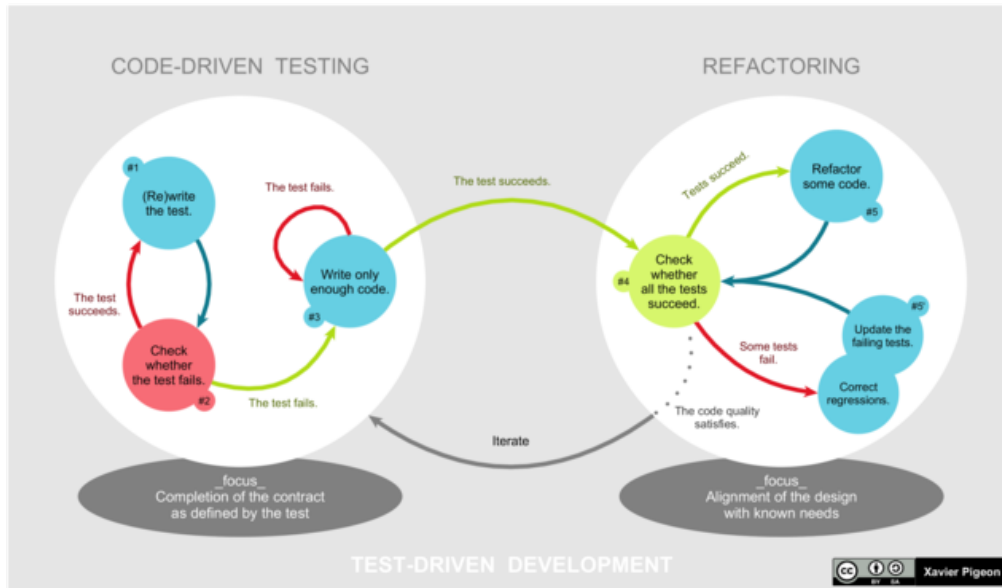


Figura 4.1: Ciclo de desarrollo TDD

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

En el contexto las pruebas de software, **ISTQB**<sup>1</sup> 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 [14].

ATDD es una extensión de TDD 4.4 en la que los tests se orientan a las pruebas de aceptación a realizar.

## 4.6. 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 [15].

<sup>1</sup>International Software Testing Qualifications Board

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 instintiva, 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 4.2 [16] muestra el ciclo de vida del desarrollo BDD.



Figura 4.2: Ciclo de vida BDD

La Figura 4.3 [15] muestra la relación de los ciclos de desarrollo BDD y TDD.

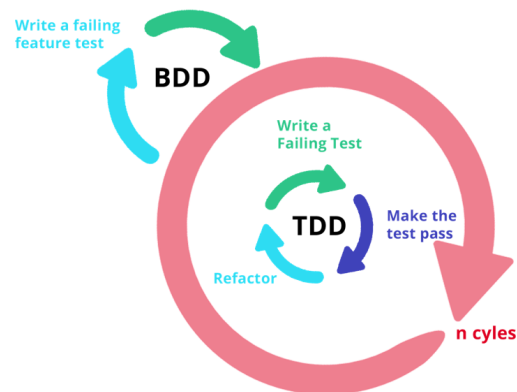


Figura 4.3: Relación de los ciclos de vida de desarrollo BDD y TDD

## 4.7. Cucumber

Ejemplo de importación de código en 4.1.

**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		
	17	7	4	2	4	2
	0	17				
Z	davidmigloz	go-bees	2017-02-01T07:00:00Z	2017-02-02T08:00:00		
	9	2	1	1	5	3
	1	9				
Z	davidmigloz	go-bees	2017-02-07T07:00:00Z	2017-02-08T08:00:00		
	7	4	0	1	2	1
	2	7				
Z	davidmigloz	go-bees	2017-02-15T07:00:00Z	2017-02-16T08:00:00		
	5	4	1	0	0	3
	1	5				

Fragmento de código 4.1: Gherkin sample code



---

# Índice de fragmentos de código

---

4.1. Gherkin sample code . . . . .	15
4.2. Clase Java convencional . . . . .	18
4.3. Clase Java utilizando Lombok . . . . .	19

## 4.8. Domain driven design (DDD)

## 4.9. Arquitectura hexagonal

TBD.

### Dominio.

.

TBD.

## 4.10. Integración continua

TBD.

## 4.11. Insomnia

## 4.12. Pipeline

TBD.

### 4.13. Open Feign

TBD.

### 4.14. Sonarqube

TBD.

### 4.15. Lombok

**Lombok**<sup>2</sup> 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 4.2.

```
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;  
    }  
}
```

---

<sup>2</sup>Project Lombok: <https://projectlombok.org/>



```

    }

    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; }

    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 4.2: Clase Java convencional

Con lombok, el código fuente quedará como el fragmento de código [4.3](#).

```

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 4.3: Clase Java utilizando Lombok

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



---

# Aspectos relevantes del desarrollo del proyecto

---

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros<sup>3</sup>, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

## 5.1. Obtención de información correspondiente a la gestión de tareas del proyecto

La información relativa a la gestión de tareas se encuentra contenida en su mayoría en GitHub, con la excepción de los puntos de historia de usuario,

almacenados en Zenhub. Para poder acceder a esta información, debemos interactuar con ambas plataformas.

## 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, el API REST ofrece un mayor ecosistema de técnicas y herramientas que facilitarán el desarrollo de la interacción.

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 peticiones**<sup>3</sup>:

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

---

<sup>3</sup>*GitHub API limit rates*: <https://docs.github.com/en/rest/overview/resources-in-the-rest-api>

## 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 Java**<sup>4</sup>
- Desarrollo de un cliente a medida.

Dado que el conjunto de operaciones es reducido - tres operaciones en el caso de la obtención de información de la gestión de tareas del proyecto, consideramos excesivo incorporar una librería para este propósito, y optamos en su lugar por desarrollar un cliente a medida mediante Open Feign 4.13.

## Pruebas de integración del cliente GitHub a medida

Para asegurarnos que la interacción con GitHub relativa a la lectura de información de la gestión de tareas es válida, planteamos una serie de pruebas automáticas mediante Cucumber 4.7, [?] reflejada 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.

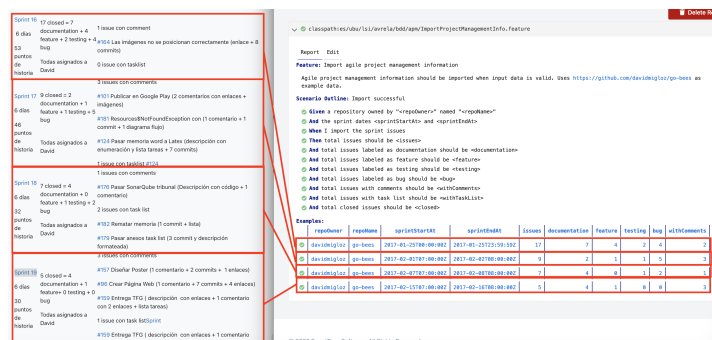


Figura 5.1: Comparativa de la información de gestión de tareas obtenida de forma manual y automática

## Interacción 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

<sup>4</sup>GitHub API for Java: <https://github-api.kohsuke.org/>

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 autenticación 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 Zenhub**<sup>5</sup> hasta revisar con el tutor las alternativas posibles.

---

<sup>5</sup>Tarea de integración con Zenhub: <https://github.com/lfx1001/avrela/issues/>

---

## **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

---

TBD Conclusiones.

Se plantean las siguientes líneas de trabajo futuras:

1. Ampliar la información a evaluar en las simulaciones . Esto conllevará ampliar el modelo de dominio así como los datos obtenidos en las integraciones con GitHub y ZenHub:
  - a)* Wiki.
  - b)* Pull requests.
  - c)* Estimación de tipo póker.



---

## 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](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, “Unit testing,” 2022, [Internet; consultado 12-noviembre-2022]. [Online]. Available: [https://en.wikipedia.org/wiki/Unit\\_testing](https://en.wikipedia.org/wiki/Unit_testing)

- [9] —, “Mock object,” 2022, [Internet; consultado 12-noviembre-2022]. [Online]. Available: [https://en.wikipedia.org/wiki/Mock\\_object#Mocks.2C\\_fakes.2C\\_and\\_stubs](https://en.wikipedia.org/wiki/Mock_object#Mocks.2C_fakes.2C_and_stubs)
- [10] —, “Integration testing,” 2022, [Internet; consultado 12-noviembre-2022]. [Online]. Available: [https://en.wikipedia.org/wiki/Integration\\_testing](https://en.wikipedia.org/wiki/Integration_testing)
- [11] —, “Test-driven development,” 2022, [Internet; consultado 12-noviembre-2022]. [Online]. Available: [https://en.wikipedia.org/wiki/Test-driven\\_development](https://en.wikipedia.org/wiki/Test-driven_development)
- [12] K. Beck, *Test-Driven Development by Example*. Addison Wesley, 2002.
- [13] 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](https://upload.wikimedia.org/wikipedia/commons/0/0b/TDD_Global_Lifecycle.png)
- [14] Wikipedia, “Acceptance testing,” 2022, [Internet; consultado 13-noviembre-2022]. [Online]. Available: [https://en.wikipedia.org/wiki/Acceptance\\_testing](https://en.wikipedia.org/wiki/Acceptance_testing)
- [15] Digité, “Behavior driven development (bdd): Creating useful software customers want,” 2022, [Internet; consultado 13-noviembre-2022]. [Online]. Available: <https://www.digite.com/agile/behavior-driven-development-bdd/>
- [16] T. point, “Cucumber - overview,” 2022, [Internet; consultado 14-noviembre-2022]. [Online]. Available: [https://www.tutorialspoint.com/cucumber/cucumber\\_overview.htm](https://www.tutorialspoint.com/cucumber/cucumber_overview.htm)