



**FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS**

**ISWZ3208 - Calidad de Software**

**Aplicación de técnicas de calidad en un proyecto**

**INFORME FINAL**

Said Carrera

Kristiany Cerón

David Navarrete

Mateo Cartagena

*Universidad de las Américas*

06/07/2025

## Informe Final

### Proceso de implementación

Inicialmente, el proyecto contaba con una clase monolítica denominada TaskManager, la cual presentaba múltiples deficiencias en cuanto a calidad de software: nombres poco descriptivos, falta de validaciones, métodos largos, ausencia de pruebas unitarias y análisis estático, entre otros.

```
import java.util.ArrayList;
import java.util.List;

public class TaskManager {
    private List tasks = new ArrayList<>();

    public void addTask(String t) {
        tasks.add(t);
        System.out.println("Task added.");
    }

    public void listTasks() {
        for (int i = 0; i < tasks.size(); i++) {
            System.out.println("Task " + (i + 1) + ": " + tasks.get(i));
        }
    }


    public void removeTask(int id) {
        tasks.remove(id - 1);
        System.out.println("Task removed.");
    }

    public static void main(String[] args) {
        TaskManager tm = new TaskManager();
        tm.addTask("Complete project");
        tm.listTasks();
        tm.removeTask(1);
    }
}
```

En base al análisis y al plan de acción definido, se llevaron a cabo diversas acciones estructuradas para mejorar el diseño, funcionalidad y mantenibilidad del sistema. A continuación, se detalla la implementación realizada:

1. **Reestructuración del sistema (Aplicación de SRP y refactorización):** Se aplicó el Principio de Responsabilidad Única (SRP) separando las funcionalidades en clases con responsabilidades claramente definidas:
  - a. **Main.java:** Punto de entrada del programa, que delega la ejecución a la clase de consola.
  - b. **Task.java:** Nueva clase encargada de representar la entidad de negocio Task, con atributos encapsulados (id, description, completed) y métodos específicos para modificar su estado (markAsCompleted, markAsIncomplete).
  - c. **TaskService.java:** Clase encargada de gestionar la lógica de negocio relacionada a las tareas. Incluye validaciones de entrada, control de ID y operaciones CRUD.
  - d. **ConsoleApp.java:** Interfaz de usuario vía consola, que se encarga de interactuar con el usuario y presentar el menú de operaciones.

Esta reestructuración resolvió los problemas relacionados con nombres confusos, métodos largos y falta de validaciones.

- 
2. **Validación de entradas y control de errores:** Se agregó validación en la creación de tareas, lanzando excepciones si la descripción es nula o vacía.
    - a. Se incorporó control para opciones inválidas del menú.
    - b. Se mejoró la eliminación de tareas añadiendo confirmación por parte del usuario.

Estas validaciones permiten un comportamiento más robusto del sistema y una mejor experiencia para el usuario.

3. **Cobertura de pruebas unitarias (JUnit 5):** Se implementaron pruebas unitarias para cubrir todos los componentes del sistema:
  - a. **TaskTest.java:** Valida constructores, getters, setters, métodos de estado, equals, hashCode y toString de la clase Task.
  - b. **TaskServiceTest.java:** Cubre los casos de creación, eliminación, búsqueda y validaciones de tareas, así como la generación de IDs únicos.
  - c. **ConsoleAppTest.java:** Simula interacciones del usuario en consola, como crear tareas, listar cuando no hay tareas o ingresar opciones inválidas.

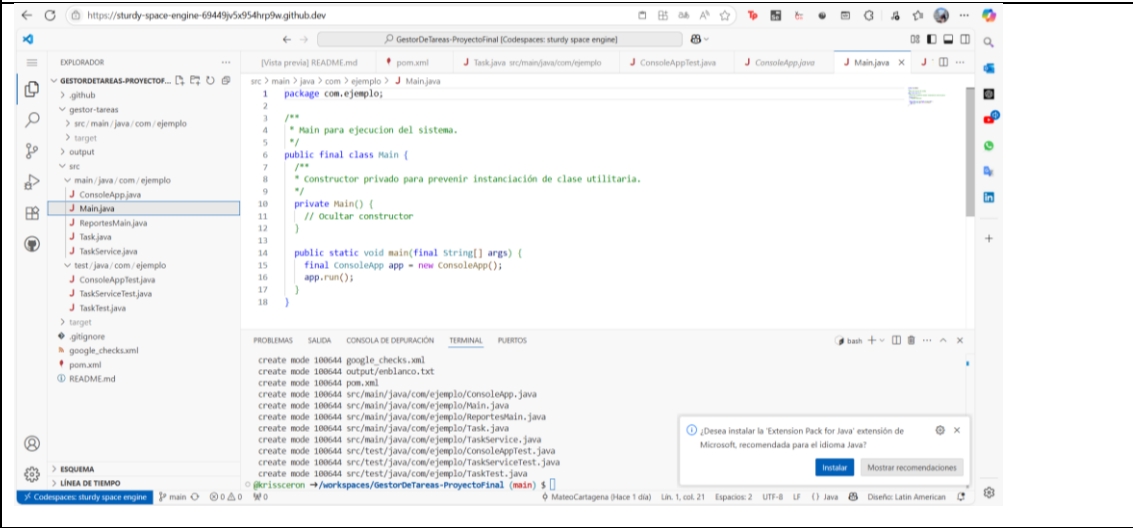
Esto permitió cumplir con la meta de un porcentaje mayor o igual al 80% de pruebas implementadas, reduciendo riesgos de regresiones.

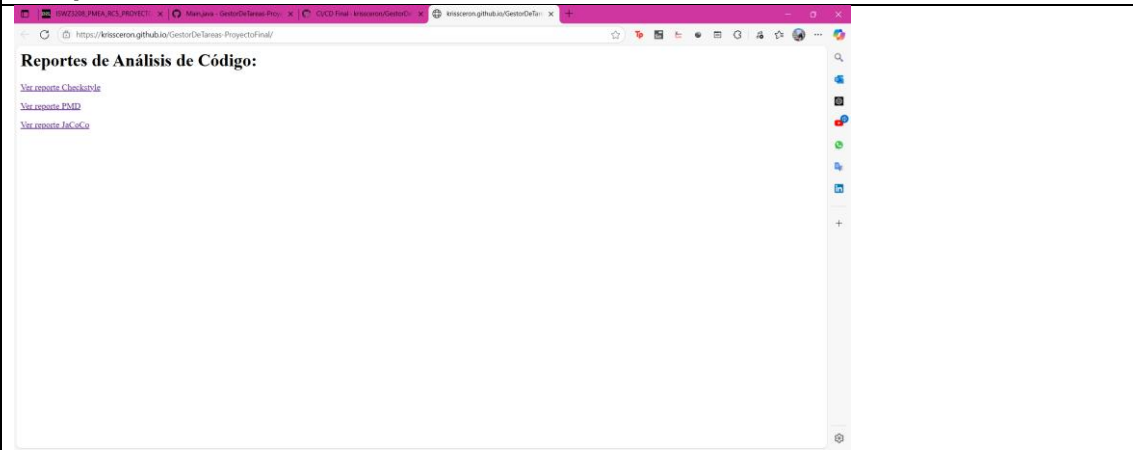
4. **Análisis estático y estilo de código:** Se configuraron y ejecutaron herramientas como:
  - a. **Checkstyle:** Para asegurar que el código siga estándares de estilo definidos.
  - b. **PMD:** Para identificar errores comunes y violaciones de diseño.
  - c. **JaCoCo:** Para medir el porcentaje de cobertura de pruebas unitarias.

Esto resolvió los problemas relacionados con la ausencia de análisis estático, asegurando un código más limpio, mantenible y libre de defectos comunes.

5. **Automatización y mejora continua (GitHub Actions):** Se integró GitHub Actions como pipeline de CI/CD para que, en cada push, se ejecuten automáticamente:
  - a. Compilación del proyecto.
  - b. Ejecución de pruebas.
  - c. Análisis de cobertura.
  - d. Verificación de estilo de código.

# Comparativa del proyecto antes y después


Código
<div>Antes</div> <pre>import java.util.ArrayList; import java.util.List;  public class TaskManager {     private List tasks = new ArrayList&lt;&gt;();      public void addTask(String t) {         tasks.add(t);         System.out.println("Task added.");     }      public void listTasks() {         for (int i = 0; i &lt; tasks.size(); i++) {             System.out.println("Task " + (i + 1) + ": " + tasks.get(i));         }     }      public void removeTask(int id) {         tasks.remove(id - 1);         System.out.println("Task removed.");     }      public static void main(String[] args) {         TaskManager tm = new TaskManager();         tm.addTask("Complete project");         tm.listTasks();         tm.removeTask(1);     } }</pre>
<div>Después</div> 


Reportes de análisis de código (Interfaz)
<div>Antes</div> <div>No existía</div>
<div>Después</div> 

## Reporte Checkstyle

### Antes

Last Published: 2025-07-06 | Version: 1.0-SNAPSHOT

 **Checkstyle Results**

The following document contains the results of Checkstyle 8.45.1 with google\_checks.xml ruleset. 

**Summary**

Files	Info	Warnings	Errors
5	0	76	0

**Files**

File	I	W	E
com/ejemplo/ConsoleApp.java	0	64	0
com/ejemplo/Main.java	0	1	0
com/ejemplo/ReportesMain.java	0	2	0
com/ejemplo/Task.java	0	6	0
com/ejemplo/TaskService.java	0	3	0

### Después

Last Published: 2025-07-06 | Version: 1.0-SNAPSHOT

 **Checkstyle Results**

The following document contains the results of Checkstyle 8.45.1 with google\_checks.xml ruleset. 

**Summary**

Files	Info	Warnings	Errors
5	0	3	0

**Files**

File	I	W	E
com/ejemplo/ReportesMain.java	0	1	0
com/ejemplo/Task.java	0	1	0
com/ejemplo/TaskService.java	0	1	0


**Rules**

Category	Rule	Violations	Severity
Javadoc	RequireEmptyLineBeforeBlockTagGroup	3	Warning

## Reporte PMD

### Antes

Last Published: 2025-07-05 | Version: 1.0-SNAPSHOT

 **PMD Results**

The following document contains the results of PMD 6.55.0.

**Violations By Priority**

Priority 3

**com/ejemplo/Task.java**

Rule	Violation	Line
UnnecessaryModifier	Unnecessary modifier 'final' on method 'getId': the method is already in a final class	23-25
UnnecessaryModifier	Unnecessary modifier 'final' on method 'getDescription': the method is already in a final class	27-29
UnnecessaryModifier	Unnecessary modifier 'final' on method 'isCompleted': the method is already in a final class	31-33
UnnecessaryModifier	Unnecessary modifier 'final' on method 'setDescription': the method is already in a final class	35-37
UnnecessaryModifier	Unnecessary modifier 'final' on method 'setCompleted': the method is already in a final class	39-41
UnnecessaryModifier	Unnecessary modifier 'final' on method 'markAsCompleted': the method is already in a final class	43-45
UnnecessaryModifier	Unnecessary modifier 'final' on method 'markAsIncomplete': the method is already in a final class	47-49
UnnecessaryModifier	Unnecessary modifier 'final' on method 'equals': the method is already in a final class	52-61
UnnecessaryModifier	Unnecessary modifier 'final' on method 'hashCode': the method is already in a final class	64-66
UnnecessaryModifier	Unnecessary modifier 'final' on method 'toString': the method is already in a final class	69-75

Priority 4

**com/ejemplo/Main.java**

Rule	Violation	Line
UnnecessaryImport	Unused import 'java.io.*'	3

### Después

Last Published: 2025-07-05 | Version: 1.0-SNAPSHOT

 **PMD Results**

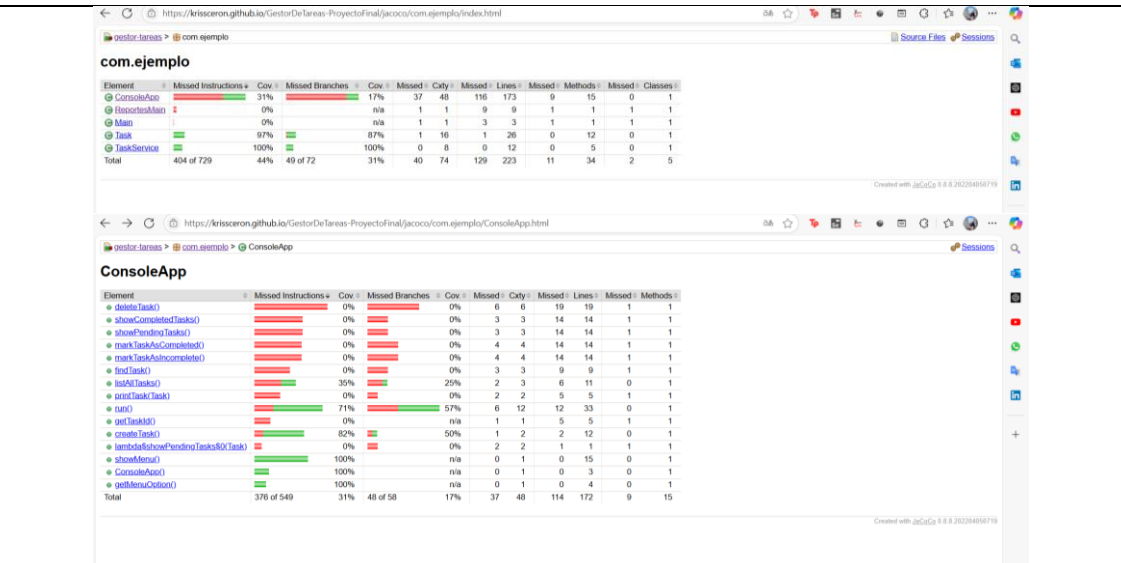
The following document contains the results of PMD 6.55.0.

PMD found no problems in your source code.

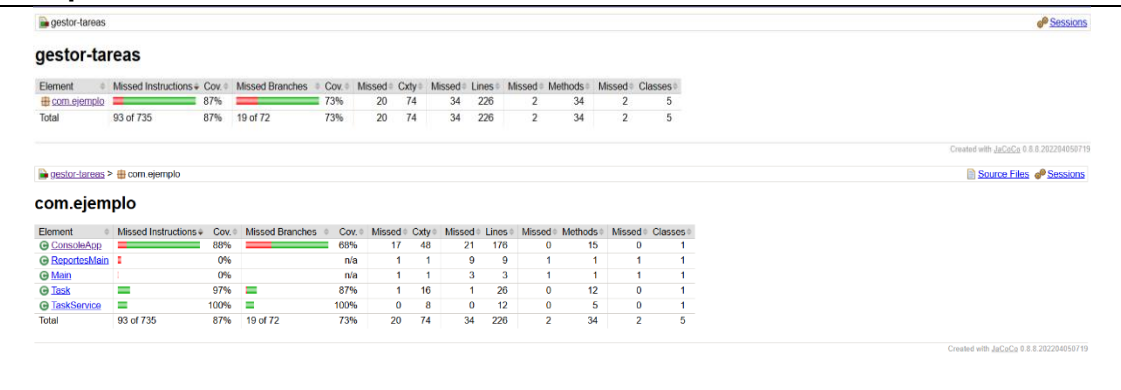
Copyright © 2025.

## Reporte JaCoCo

### Antes



### Después



## Reflexión individual sobre el trabajo en equipo.

### a) Kristiany Cerón

En este proyecto final no solo aplicamos los conocimientos técnicos adquiridos durante el semestre, sino también habilidades sociales clave, como el trabajo en equipo. La elaboración del plan de acción fue fundamental, ya que nos permitió definir con claridad los objetivos, identificar las deficiencias del código original, analizarlas en profundidad y proponer soluciones utilizando métricas específicas. Durante el desarrollo, implementamos los principios SOLID necesarios para mejorar la estructura del código, realizamos pruebas unitarias y llevamos a cabo un análisis estático con herramientas como JaCoCo, Checkstyle y PMD. Finalmente, aunque enfrentamos algunos retos al implementar la integración y entrega continuas (CI/CD), logramos resolverlos en equipo y concretar exitosamente esta automatización.



## b) David Navarrete

Este proyecto nos permitió aplicar de manera práctica los conocimientos adquiridos en la materia de Calidad de Software, fortaleciendo tanto nuestras habilidades técnicas como colaborativas. A través de la implementación de principios SOLID (especialmente SRP), pruebas unitarias con JUnit 5 y herramientas de análisis como Checkstyle, PMD y JaCoCo, mejoramos la estructura, legibilidad y mantenibilidad del sistema base. Así también, la integración del pipeline CI/CD con GitHub Actions nos acercó a prácticas más actuales del desarrollo profesional. Por lo tanto, el trabajo en equipo fue clave para distribuir tareas, tomar decisiones conjuntas y experimentar un flujo de desarrollo organizado, lo que enriqueció nuestra formación académica y nuestra preparación para escenarios reales en el ámbito del software.

## c) Said Carrera

Durante este proyecto entendí que la calidad del software va más allá de que un programa simplemente “funcione”. Invertir tiempo en aplicar buenas prácticas, como la separación de responsabilidades, la validación de entradas y la implementación de pruebas automatizadas, nos permite construir soluciones sostenibles y mantenibles a largo plazo. Enfrentamos desafíos en una parte del proyecto, pero el trabajo en equipo y la organización fueron clave para superarlos. Personalmente, este proyecto me ayudó a valorar la importancia del control de versiones, el análisis estático y la integración continua como parte de un proceso profesional de desarrollo.

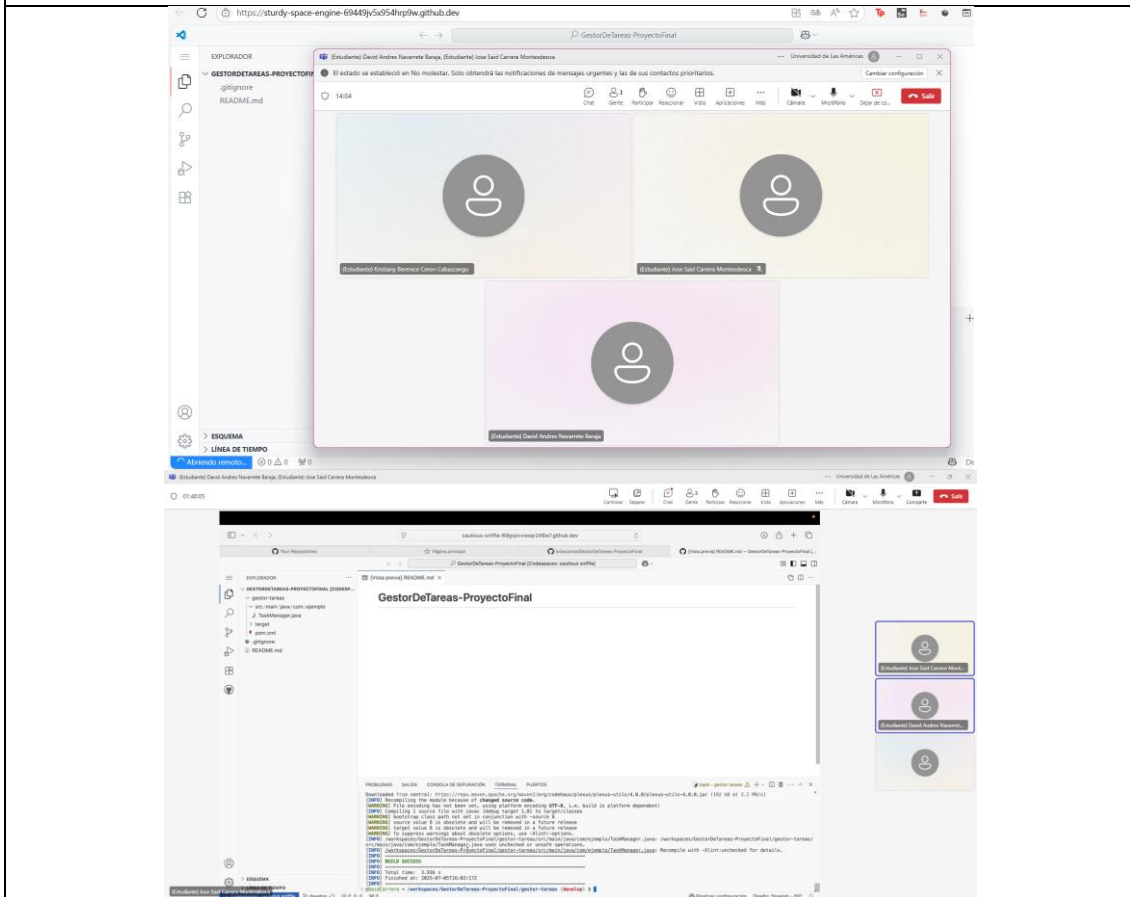
## d) Mateo Cartagena

Parecía cosa de nada, un pequeño código de no más de 50 líneas que tenía una funcionalidad clara y que, de cierta manera, cumplía con su propósito. Siempre se comenta en la ingeniería que existe una máxima, y esta es: “si funciona, no lo toques”. Sin embargo, como ingenieros y estudiantes próximos a ser unos profesionales, creo que deberías entender qué es ofrecer un trabajo de calidad. Calidad de Software nos ha enseñado herramientas y tecnologías útiles y necesarias en el mercado al que muy pronto serás parte. Trabajar en equipo, estándares, buenas prácticas, son cualidades que nunca se deberían descuidar, pues apoyan la creación de trabajos prolijos y de calidad, realizados por personas competentes.

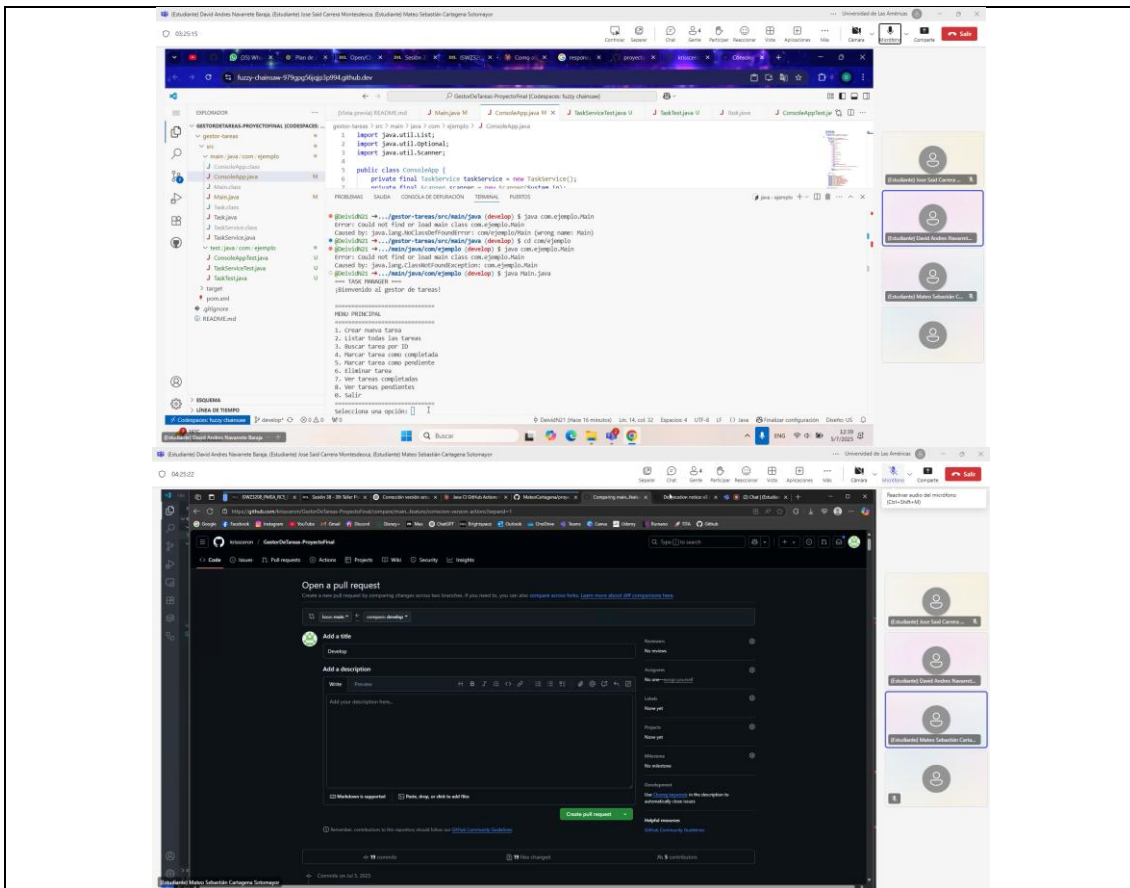
## Evidencias

- a) Evidencias para el Trabajo Colaborativo: Registros de reuniones (videos, chats, etc.) y contribuciones individuales (commits en GitHub).

### Reunión vía Teams



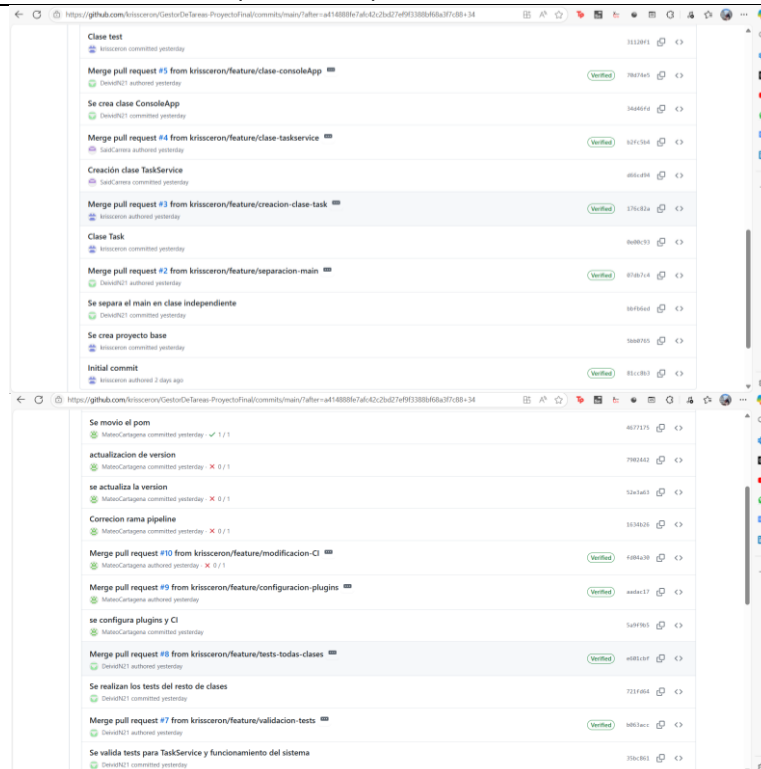




Enlace de reunión del equipo:

[Llamada con \(Estudiante\) y 2 más-20250705\\_161724-Grabación de la reunión.mp4](#)

Contribuciones Individuales (Commits):



Commits · [krissceron/GestorDeTareas-ProyectoFinal](#)

## CI/CD Funcional

krissceron / GestorDeTareas-ProyectoFinal

<> Code Issues Pull requests **Actions** Projects Wiki Security Insights

← Java CI Build and Quality Analysis

**CI/CD Final #45**

**Summary**

Jobs

- build-and-analyze
- deploy

Run details

- Usage
- Workflow file

Triggered via push 4 minutes ago

DeividN21 pushed → #141688 main

Status	Total duration	Artifacts
Success	51s	4

**ci-pipeline.yml**  
on: push

```

graph LR
    A[build-and-analyze 35s] --> B[deploy 11s]
    B --> C[https://krissceron.github.io/GestorDeTareas...]
  
```

## Ejecución de Sistema

The screenshot shows an IDE with two main panes. The left pane displays a project structure for 'test.java.com.gemelo' containing files like 'ConsoleApp.java', 'Main.java', 'Reportas.java', 'Task.java', and 'Usuario.java'. Below this is a 'target' directory with files like 'gigloone', 'google-checkout', 'pom.xml', and 'README.md'. The right pane shows the terminal output of a Maven command. The output indicates that the module is being recompiled due to changes in source code, and it lists several system modules that are not set in conjunction with source 11. The terminal also shows the execution of a Java command to run the application.

**CRUD de Gestor de Tareas (Crear – Listar – Actualizar - Eliminar):**

```

1 package com.ejemplo
2
3 import java.util.*;
4
5 public class Main {
6     public static void main(String[] args) {
7         ConsoleApp app = new ConsoleApp();
8         app.run();
9     }
10 }
11
12 // ... (other code) ...
13
14 // ... (other code) ...
15
16 // ... (other code) ...
17
18 // ... (other code) ...
19
20 // ... (other code) ...
21
22 // ... (other code) ...
23
24 // ... (other code) ...
25
26 // ... (other code) ...
27
28 // ... (other code) ...
29
30 // ... (other code) ...
31
32 // ... (other code) ...
33
34 // ... (other code) ...
35
36 // ... (other code) ...
37
38 // ... (other code) ...
39
40 // ... (other code) ...
41
42 // ... (other code) ...
43
44 // ... (other code) ...
45
46 // ... (other code) ...
47
48 // ... (other code) ...
49
50 // ... (other code) ...
51
52 // ... (other code) ...
53
54 // ... (other code) ...
55
56 // ... (other code) ...
57
58 // ... (other code) ...
59
60 // ... (other code) ...
61
62 // ... (other code) ...
63
64 // ... (other code) ...
65
66 // ... (other code) ...
67
68 // ... (other code) ...
69
70 // ... (other code) ...
71
72 // ... (other code) ...
73
74 // ... (other code) ...
75
76 // ... (other code) ...
77
78 // ... (other code) ...
79
80 // ... (other code) ...
81
82 // ... (other code) ...
83
84 // ... (other code) ...
85
86 // ... (other code) ...
87
88 // ... (other code) ...
89
90 // ... (other code) ...
91
92 // ... (other code) ...
93
94 // ... (other code) ...
95
96 // ... (other code) ...
97
98 // ... (other code) ...
99
100 // ... (other code) ...

```