Facultad de Ingeniería y Ciencias Aplicadas, Universidad de las Américas (UDLA).

ISWZ3208 (5582)- Calidad de Software.

Docente: Guillermo Alfredo Ávila Noboa.

18 de Enero de 2026.

Plan de acción del Proyecto Integrador.

Integrantes del grupo.

Jossue Enrique Ayala Farfán.

Esteban Manuel Carvajal Landázuri.

Jorge Nicolas Negrete Viteri.

Ricardo Josué Pérez Jérez.

Diego Josué Vega Aguilera.

I Introducción.

El presente Plan de Acción tiene como finalidad organizar y guiar la aplicación de técnicas de calidad de software en un proyecto Java, haciendo uso de principios de Crean Code, métricas de calidad y herramientas de análisis automatizado. El proyecto seleccionado como base es un Sistema de Gestión de Tareas, el cual permite crear, listar y eliminar tareas, y que será extendido para incluir la funcionalidad de actualización.

Este proyecto integrador parte de un código inicial con problemas de calidad predefinidos, tales como nombres poco descriptivos, baja cohesión, ausencia de validaciones, cobertura insuficiente de pruebas unitarias y falta de análisis automatizado. Estas deficiencias lo convierten en un caso adecuado para aplicar de manera práctica los conceptos revisados en la asignatura y demostrar mejoras medibles y verificables.

II Roles y responsabilidades del equipo.

Para garantizar una ejecución ordenada del proyecto y fomentar el trabajo colaborativo, el equipo definió roles específicos alineados con las actividades técnicas requeridas y con las habilidades individuales de cada integrante. Esta organización permite distribuir responsabilidades de manera equilibrada y asegurar la integración de todas las fases del proceso de mejora de calidad.

II.2 Roles asignados.

De esta manera, los roles definidos en base a lo que se requiere en el proyecto, quedan de la siguiente manera:

II.2 Responsabilidades para cada integrante.

En consenso con todos los integrantes, y con los roles definidos para el proyecto, se delegan las siguientes tareas para la ejecución del mismo según el rol.

Líder del Proyecto – Esteban Carvajal
Tiene como objetivo principal coordinar el trabajo del equipo y supervisar que la implementación de mejoras se ejecute conforme al Plan de Acción. Sus responsabilidades incluyen el seguimiento del avance general, la validación de la integración del código en el repositorio GitHub y la verificación de que las pruebas unitarias y el pipeline CI/CD funcionen correctamente. Este rol es clave para mantener coherencia entre las decisiones técnicas y lo que se quiere lograr en el proyecto.

Analista de Métricas – Ricardo Pérez
Se encarga de definir y medir indicadores cuantitativos que permitan evaluar la calidad del software antes y después de la refactorización. Su trabajo consiste en configurar las herramientas de medición, establecer una línea base inicial y preparar la estructura comparativa que será utilizada en el Informe Final. Como resultado de esta fase, ya se cuenta con métricas reales de cobertura, estilo y defectos, obtenidas mediante herramientas automáticas.

Responsable de Revisión Manual – Diego Vega
Tiene como objetivo mejorar la calidad interna del código mediante la aplicación de principios de Clean Code y SOLID. Este rol se enfoca en la revisión línea por línea del código, la refactorización para mejorar legibilidad y cohesión, y la incorporación de validaciones básicas que aumenten la robustez del sistema. El diagnóstico inicial del código permitió identificar problemas estructurales y funcionales que servirán como base para las mejoras posteriores.

Responsable de Análisis Estático – Jorge Negrete
Es el encargado de configurar y ejecutar herramientas automáticas de análisis de código, tales como Checkstyle, PMD y SpotBugs. Además, integra estas herramientas

dentro del pipeline de GitHub Actions, permitiendo que el análisis de calidad se ejecute automáticamente en cada cambio del repositorio y asegurando el cumplimiento continuo de los estándares definidos.

Responsable de Documentación y Revisión – Jossue Ayala	
Se enfoca en consolidar y presentar de manera clara y estructurada el trabajo técnico del equipo. Este rol incluye la elaboración del Plan de Acción, el Informe Final y la presentación grupal, así como la recopilación de evidencias del trabajo colaborativo, métricas obtenidas y funcionamiento del pipeline CI/CD.	

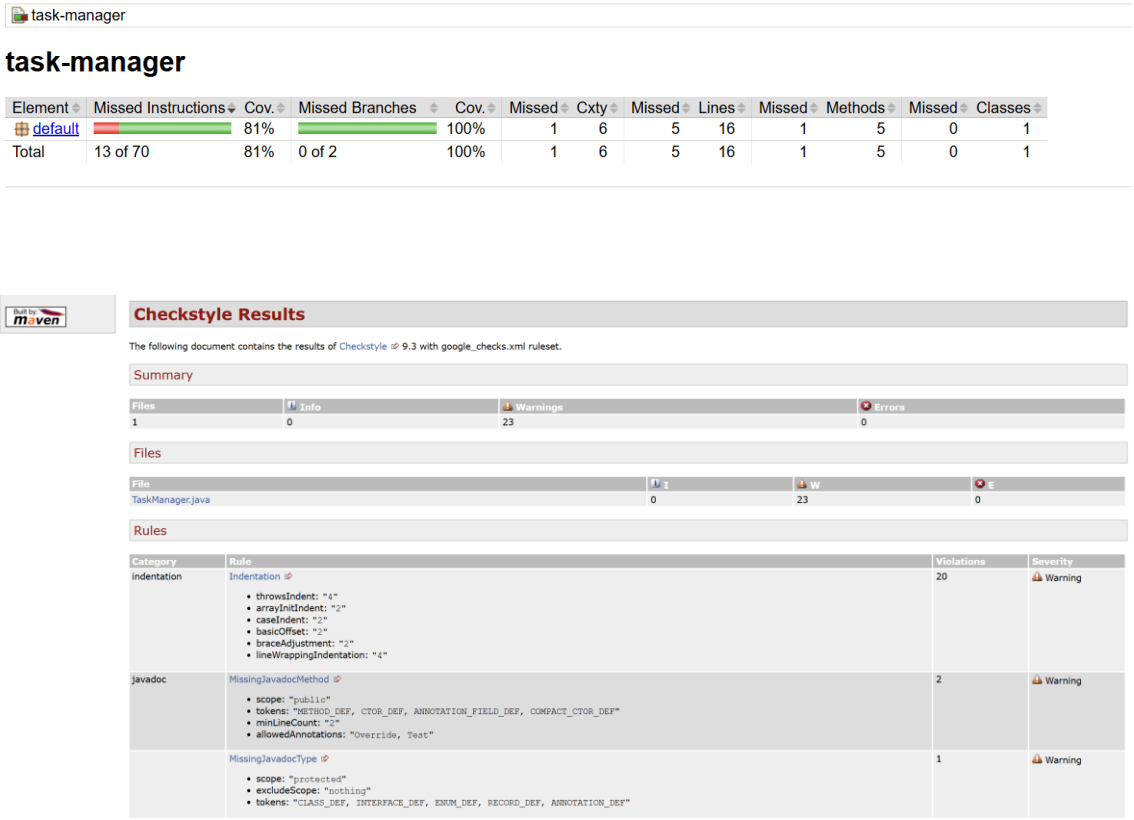
III Problemas de calidad identificados.

A partir de la revisión inicial del proyecto base, se identificaron diversos problemas de calidad que afectan la mantenibilidad, legibilidad y robustez del sistema. En primer lugar, se detectaron problemas de legibilidad, como el uso de nombres poco descriptivos en variables y parámetros, por ejemplo ***t y tm***, así como el uso de colecciones sin genéricos, lo que reduce la seguridad de tipos y dificulta la comprensión del código

Asimismo, el código presenta baja cohesión, ya que los métodos encargados de la lógica de negocio también realizan tareas de salida por consola mediante ***System.out.println***. Esta mezcla de responsabilidades incrementa el acoplamiento y dificulta la realización de pruebas unitarias efectivas, afectando directamente la calidad interna del software.

Otro problema relevante es la ausencia de manejo adecuado de errores y validaciones. El método encargado de eliminar tareas no valida si el identificador recibido se encuentra dentro de un rango válido, lo que puede provocar errores en tiempo de ejecución. De igual forma, el sistema permite agregar tareas nulas, vacías o duplicadas, incumpliendo los requerimientos establecidos en el enunciado del proyecto.

En cuanto a métricas, el estado inicial del proyecto carecía de pruebas unitarias y análisis automatizado. Sin embargo, como parte del trabajo inicial, se estableció una línea base cuantitativa, obteniendo resultados medibles mediante herramientas vistas en el semestre: 23 advertencias de estilo detectadas por Checkstyle, 0 violaciones PMD y una cobertura de pruebas del 81% a nivel de instrucciones y 100% a nivel de ramas con JaCoCo.



task-manager

task-manager

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
default		81%		100%	1	6	5	16	1	5	0	1
Total	13 of 70	81%	0 of 2	100%	1	6	5	16	1	5	0	1

Checkstyle Results

The following document contains the results of Checkstyle 9.3 with google_checks.xml ruleset.

Summary

Files	Info	Warnings	Errors
1	0	23	0

Files

File	I	W	E
TaskManager.java	0	23	0

Rules

Category	Rule	Violations	Severity
indentation	Indentation • throwsIndent: "4" • arrayInitIndent: "2" • caseIndent: "2" • basicOffset: "2" • braceAdjustment: "2" • lineWrappingIndentation: "4"	20	Warning
javadoc	MissingJavadocMethod • scope: "public" • tokens: "METHOD_DEF, CTOR_DEF, ANNOTATION_FIELD_DEF, COMPACT_CTOR_DEF" • minLineCount: "2" • allowedAnnotations: "Override, Test"	2	Warning
	MissingJavadocType • scope: "protected" • excludeScope: "nothing" • tokens: "CLASS_DEF, INTERFACE_DEF, ENUM_DEF, RECORD_DEF, ANNOTATION_DEF"	1	Warning

Capturas obtenidas de los resultados del proyecto base (Con Checkstyle y JaCoCo).

IV. Estrategias y métricas propuestas.

Con base en los problemas identificados, se definieron estrategias de mejora enfocadas en Clean Code, principios SOLID y automatización del control de calidad. En primer lugar, se plantea mejorar la legibilidad y mantenibilidad del código, renombrando variables y métodos con nombres descriptivos y utilizando genéricos en las colecciones. Esta estrategia busca que el código sea comprensible sin necesidad de comentarios adicionales.

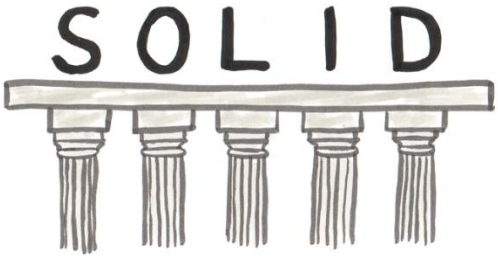
En segundo lugar, se propone una separación clara de responsabilidades, eliminando la impresión por consola dentro de los métodos de lógica de negocio y centralizando dicha funcionalidad en el método principal. Esto permitirá que la lógica del sistema sea fácilmente testeable y cumpla con el Principio de Responsabilidad Única.

Otra estrategia clave es la incorporación de validaciones y manejo básico de errores, evitando la inserción de tareas inválidas o duplicadas y controlando adecuadamente los identificadores utilizados para eliminar o actualizar tareas. Además, se contempla completar la funcionalidad del sistema mediante la implementación de la operación de actualización de tareas, alineando el código con la descripción funcional del

proyecto.

Para evaluar objetivamente el impacto de estas mejoras, se utilizarán métricas automáticas de calidad. La cobertura de pruebas será medida mediante JaCoCo, considerando el porcentaje de instrucciones y ramas ejecutadas. El estilo del código será evaluado con Checkstyle, contabilizando el número de violaciones detectadas, mientras que los defectos y malas prácticas serán identificados mediante PMD. Estas métricas permitirán realizar una comparación clara entre el estado inicial y el estado final del proyecto.

Finalmente, todas las herramientas de análisis serán integradas en un pipeline CI/CD con GitHub Actions, garantizando que las métricas y reportes se generen automáticamente y sirvan como evidencia del cumplimiento de los objetivos de calidad definidos.



IV. Estrategias y métricas propuestas.

El presente Plan de Acción define una estructura clara para la aplicación de técnicas de calidad de software en el proyecto *TaskManager* de Java a ser ejecutado, estableciendo roles justificados, identificando problemas concretos de calidad y proponiendo estrategias de mejora respaldadas por métricas objetivas y herramientas automatizadas. Esta planificación permite abordar la calidad del software de forma organizada y colaborativa.

A partir de este plan, el *Informe Final* deberá evidenciar las mejoras implementadas mediante una comparación clara entre el estado inicial y final del proyecto, utilizando métricas de cobertura, estilo y defectos, así como reflejar el proceso de refactorización, automatización y trabajo en equipo desarrollado durante el proyecto.